Math

COO-1018-1206

STANDARDIZATION OF CONTROL POINT REALIZATION

by

Ronald Gustav Martin

May 21, 1970

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

COO-1018-1206

Report No. 400

STANDARDIZATION OF CONTROL POINT REALIZATION*

by

Ronald Gustav Martin

May, 1970

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

*Supported in part by Contract Number U.S. AEC AT(11-1)-1018 and submitted in partial fulfillment for the degree of Masters of Science in Electrical Engineering, at the University of Illinois, May, 1970.

ACKNOWLEDGEMENT

The author would like to thank his advisor, Professor Sylvian R. Ray, for his helpful advice and guidance in the preparation of this thesis.

The author would also like to thank Mrs. Betty Gunsalus for typing the final draft and Mr. Stanley Zundo for preparing the figures included in this thesis and finally to Mr. Dennis Reed and the offset department for the reproduction of this thesis.

TABLE OF CONTENTS

Page

INTRODUCTION..... 1

1. PSEUDO-ASYNCHRONOUS CONTROL DESIGN..... 2

1.1 Design Strategy..... 2

1.2 Control Reliability and Maintenance..... 4

2. FLOW CHART..... 6

2.1 Function..... 6

2.2 Notation..... 7

2.3 Examples..... 11

3. CONTROL POINT - A BUILDING BLOCK APPROACH..... 17

3.1 Description..... 17

3.1.1 Task Stage..... 18

3.1.2 Timing Stage..... 23

3.2 Sequence Stage..... 30

4. CONTROL LOGIC IMPLEMENTATION..... 38

4.1 Conversion of Flow Charts to Control Logic Representation.. 38

4.2 Hardware Realization..... 43

4.3 Question of Minimization vs. Standardization..... 45

CONCLUSION..... 47

LIST OF REFERENCES..... 48

APPENDIX

I. Examples of Actual Flow Chart to Logic Drawing Conversions.. 49

II. Test Results for Control Point Configuration..... 58

LIST OF TABLES

Table		Page
2.3.1	Mathematical Designation for MØNPAR.....	12
2.3.2	Identification of Terms for MØNPAR.....	12
2.3.3	Mathematical Designation for XSET.....	14
2.3.4	Identification of Terms for XSET.....	15
4.1.1	Conversion of Flow Chart to Logic Representation...	39

LIST OF FIGURES

Figure		Page
2.2.1	Example of a Routine Symbol	9
2.2.2	Example of a Task Box	9
2.2.3	Example of Parallel Conditional Tasks	9
2.2.4	Examples of Decision Symbol	10
2.2.5	Example of Reply Symbol	10
2.3.1	Flow Chart of MØNPAR	13
2.3.2	Flow Chart of XSET	16
3.1.1.1	Block Diagram of Task Stage Logic	19
3.1.1.2	Most Elementary Task Stage Configuration	20
3.1.1.3	Multiple "Advance In" Inputs to Memory Element	22
3.1.2.1	Timing Stage Using Internal Delay Element	24
3.1.2.2	Timing Stage Equivalent Circuits	26
3.1.2.3	Delay Circuit with Diode to Enhance Recovery	28
3.1.2.4	Timing Stage Using External Reply	28
3.1.2.5	Control Point Block Diagram and Circuit Configuration	29
3.2.1	Two Way Branch Sequence Stage Logic	31
3.2.2	WAIT Condition Using EN Gate	33
3.2.3	WAIT Condition Using $\overline{A_o}$ Line	34
3.2.4	Interlocking Two Parallel Control Chains	35
3.2.5	"Calling Control Point" Circuit Configuration	37
4.1.1	Logic Drawing for MØNPAR	42
4.2.1	Control Point Card	44

INTRODUCTION

The philosophy of computer control design has generally tended to be quite fluid in its approach. Variation in design approaches can be attributed to such factors as: control compatibility with computer configuration, advances in both hardware and software technology, and the consideration of speed and cost requirements. A control can be basically considered as being synchronous, asynchronous, pseudo-asynchronous, or microprogrammed and the design techniques used are a function of this classification.

A problem area common to all control design techniques has been recognized for some time; there exists a discontinuity in the communication between the procedural or software control design approach and that of the implementation of this design into hardware by the engineer.

The purpose of this paper is to specify some techniques which more closely bind together the software and hardware approaches to control design and in particular, the design of a pseudo-asynchronous type of control.

1. PSEUDO-ASYNCHRONOUS CONTROL DESIGN

1.1 Design Strategy

Historically, the University of Illinois Digital Computer Laboratory has been primarily involved in the investigation and construction of asynchronous computers. The use of asynchronous control has been dictated by the parallel operation of the various processors and the general complexity and speed considerations of the complete system. By designing the computer so that its behavior is independent of the relative speeds of the elements, one may ignore the problem of matching speeds and synchronizing signals to achieve correct operation. While asynchronous networks do have certain advantages over synchronous ones, there are some design restrictions as Braun (1963) has stated,

> "because of an indeterminate time for execution of operations and the complexities required in general for hazard free operation, asynchronous systems are considered more difficult to design, understand, and service."

It then is the responsibility of both the mathematican and the logical design engineers to work together to optimize the design of the asynchronous control such that these restrictions are minimized. Control logic can generally be considered as being either speed-dependent or speed-independent is its operation.

A speed-dependent circuit can be basically described as one in which the operation is a function of time as determined by a model flip-flop or timing delay. That is, once a control circuit is activated, there is a finite time allocated for the control to perform its various tasks. Each control function must be carefully analyzed to determine the minimum time required to perform its specific task. Consideration must be given to: the

variations of component parameters, propagation delay times, and the effects of component aging and voltage changes. While the hardware realization from logical flow charts is quite straight forward for this type of control, there are some sacrifices to both operational speed and reliability.

Speed-independent or asynchronous logic is different from speed-dependent logic in that reply signals are used to indicate the completion of a given operation. These reply signals are incorporated in the basic sequencing control logic and elimiates the need for model flip-flops or timing delays. The operational speed is a function of the time actually required to complete a specific task and there, therefore, can be considered to be optimized. While this type of control does not have the disadvantages of a speed-dependent system, it does require additional hardware which greatly increases the design complexity and renders a logical design problem of flow chart realization. Robert Swartout (1963) performed extensive studies in speed-independent logic for control and stated:

> "The majority of the logical design problems were presented as information flow charts to be realized. Unfortunately, the state of the speed-independent logical design art is still mostly an art and not a science."

The design of the control units used in the Illiac III* computer and the design techniques described in this paper are a combination of speed-dependent and speed-independent circuits (pseudo-asynchronous control) incorporating the advantages of both. Since the design of this particular control logic, like many other synthesis procedures, does not necessarily produce a unique topology, design tachniques have been developed in the Illiac III project to insure consistency throughout the control.

---

*Illiac III is an AEC funded pattern recognition computer being constructed at the University of Illinois.

1.2 Control Reliability and Maintenance

Anyone who has had the opportunity to be involved in the operation of a digital computer, will agree that it is only useful so long as it functions correctly and consistently. As digital computers have been designed to operate at faster and faster rates and have become greater in size and complexity, the requirements for malfunction-free operation by each circuit element in the machine have become extremely high. It is conceivable that circuit elements might be required to operate as many as 10^{16} times in a single day without an error. This high degree of reliability can be attained by the proper selection of components, the utilization of specific fabrication techniques, along with design strategies which have proven to function as free of malfunctions as possible. With the advances in both component and fabrication technologies, the assurance of control reliability becomes related very closely with the design techniques being employed.

No matter how much consideration is given to control reliability, there will be certain unavoidable malfunctions which do occur. With the increasing size of computing machines, it does not require a great deal of imagination to envision the problems associated with the location of a defective circuit element within a system that might employ thousands of these elements. It is, therefore, necessary that the computer control circuitry be so designed that not only are the apparently unattainable requirements for circuit reliability met, but also that it is relatively easy to service and maintain as well.

In the past, a circuit malfunction which has been detected in a processor of the computer has usually been localized by the use of diagnostic routines. If a circuit within the control malfunctions, the probability of

locating this trouble is much less than it would be for another unit or processor in the computer. This is due to the fact that the incorrectly operating circuit can cause the diagnostic routines to be improperly executed, which might just add to the total confusion that already exists. The actual maintenance or "de-bugging" of the control then becomes one that is related to how clever or experienced the service personnel are and is at best, one that is strictly on a hit and miss proposition.

The control should then be designed to operate with greater reliability and be easier to service than any other portion of the machine.

2. FLOW CHART

2.1 Function

The flow chart is the means by which the mathematician or software orientated individual conveys to the logical designer the time or sequential ordering of operations to be performed in order that a given instruction/routine, or subset of an instruction/routine, be properly executed. It can be one that contains only the essential information used in the initial planning stage of control design or it might be a detailed functional description used for check-out and maintenance. As Gillies (1961) noted:

> "As a practical matter, a flow chart notation should preferably be compact, easily drawn freehand, and should exhibit only the essential information. This is important at the planning stage because such flow charts are very susceptible to error and must be redrawn, with corrections, many times."

The flow chart is actually then a mathematical description that corresponds to a logical design and can be used to either implement this design into hardware or as a check to insure that the control design is correct.

It is essential then that the mathematicians and logical designers work closely together to insure that these flow charts are representative of a design in which there has been a minimization of the number of steps required to perform a given operation or routine. There should also be mutual agreement to such trivial details as the naming of gates, registers, buffers, and other associated pieces of hardware that will be acted upon by the control.

If the flow charts are presented to the logical designer in an optimized, final form, along with a detailed listing and description of names used in these charts, his task of implementing these flow charts into an actual logical hardware design will be greatly simplified.

2.2 Notation

There are basically only four different types of symbols used in the representation of these flow charts. There is also special consideration given to the notation contained within these symbols.

The symbol ◯ is used to indicate either the entry or call of a routine with its name appearing within the symbol and is shown in Figure 2.2.1.

Probably the most important and commonly used symbol in the flow charts is the task box ▭ . This box, as shown in Figure 2.2.2, indicates the task or operation that is to be performed in the time ordered sequence of events that occur during a given routine. Within the box is notation corresponding to the specific operation that occurs when the task box is entered. This notation follows the format XXX - Y - ZZ, where XXX is the name of the element or device upon which the task will be performed (i.e., the dependent variable), Y indicates the type of operation being executed by the box, and ZZ specifies the operational variable (i.e., the independent variable). For example, LSB/G = 1 indicates that the gate of an element LSB would be "turned on" during the operation of this particular control event. Another example is XDU ← A, where the value of a variable A, is loaded into a memory device XDU.

Associated with each task is a certain duration of time after which a reply signal is generated and control proceeds along the exit line of the box.

When more than one task can be initiated concurrently, the conditions which must exist for the task to take place are indicated above the task entry line as shown in Figure 2.2.3. The reply lines of the TASK A and TASK B boxes merge together on one horizontal line. Control will advance to the next stage of the flow chart when both replies are true, that is when Reply A • Reply B = "1".

The symbol  is used to represent a decision is to be performed and the name of the dependent variable is noted within the symbol. Figure 2.2.4 illustrates the use of this decision symbol. If DONE = "1", then go to X, whereas if DONE = "0", then control would go to Y. The decision could also be used to cause a "wait" until certain conditions are met.

The termination or reply of a given routine is indicated by the symbol  and is shown in Figure 2.2.5. The termination of a routine might be due to either an error occurring or just the normal completion of the tasks, so therefore, a memory element for either case is included with the reply signal.

The flow chart will then consist of various combinations of these four symbols connected by lines with arrowheads indicating the direction of flow. In most instances, the chart will be layed out such that the flow will be from top to bottom.

The selection of the symbols and notation described in this paper are those used by the author in the design of the Scanner-Monitor-Video Controller for the Illiac III computer and are actually quite arbitrary in nature. The specific symbolic and notational representation to use is strictly a matter of personal choice and mutual agreement between the software and hardware individual involved in the design of a control.

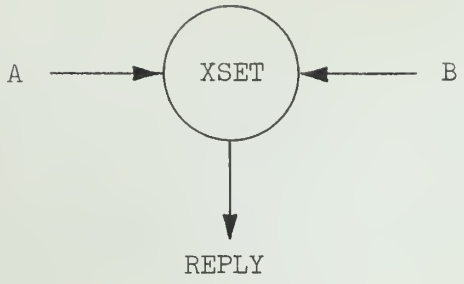


Figure 2.2.1 - Example of a Routine symbol

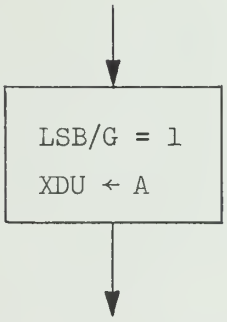


Figure 2.2.2 - Example of a Task Box

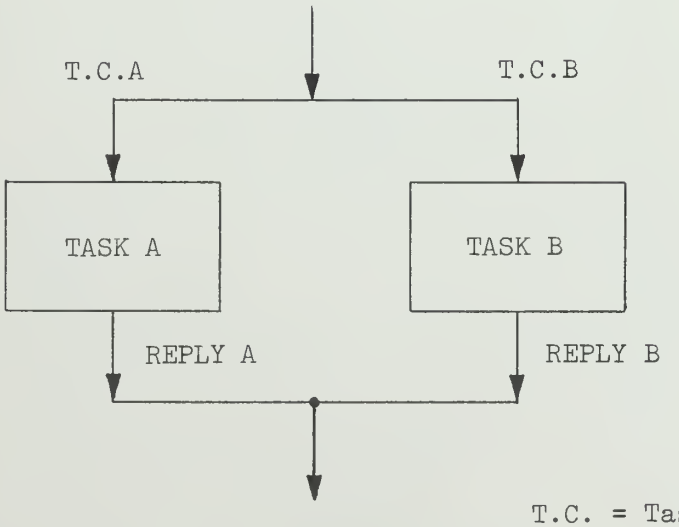


FIGURE 2.2.3 - Example of Parallel Conditional Tasks

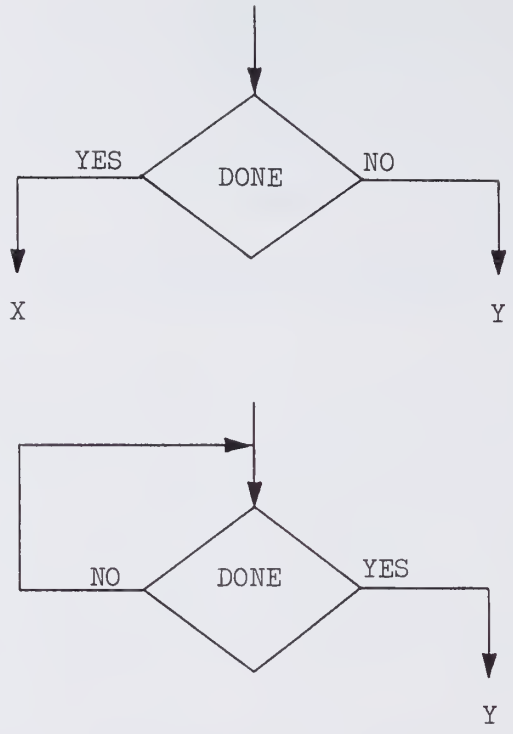


Figure 2.2.4 - Examples of Decision Symbol

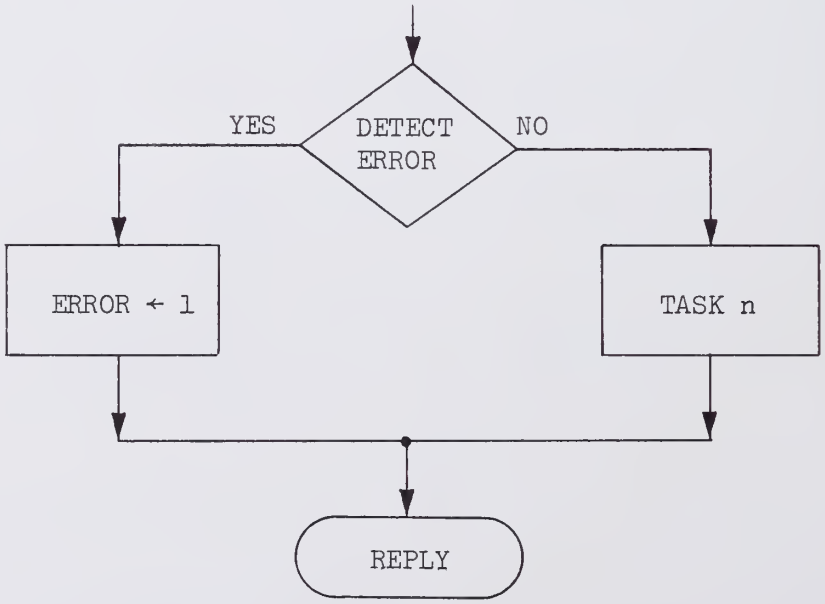


Figure 2.2.5 - Example of Reply Symbol

2.3 Examples

Several flow charts will be examined to reinforce the initial ideas and views which were presented in the proceeding section. These examples will show the mathematicians design approach, along with the identification of terms, and the flow chart equivalent.

The first example will show the hierarchy of subroutines contained within a routine called MØNPAR.* Table 2.3.1 is the mathematical design approach and the identification of terms has been listed in Table 2.3.2. The flow chart equivalent of MØNPAR can be seen in Figure 2.3.1.

The second example will deal more explicitly with one of these subroutines XSET. The mathematical approach and the identification of terms are shown in Table 2.3.3 and Table 2.3.4 respectively. Figure 2.3.2 illustrates the flow chart for this subroutine.

---

* MØNPAR is a portion of the Scanner-Monitor Communication Control used in the Illiac III computer.

MØNPAR (SET MONITOR PARAMETERS)

① Called by BMC or INCR
② Call MINIT
③ Call α SET (where α ε (W, X, Y, Z))
④ SEND = 1
⑤ If $\overline{\text{DONE}}$, go to ③
⑥ Reply

Table 2.3.1 Mathematical Designation for MØNPAR

BMC,	Beam Motion Control
INCR,	Incremental Control
MINIT,	Initialize monitor parameters routine
αSET,	(4) Set monitor parameter routines i.e., WSET, XSET, YSET, ZSET
SEND,	Flag communication bit
DONE,	Indicates all parameters are properly set

Table 2.3.2 Identification of Terms for MØNPAR

SET MONITOR PARAMETERS - (MØNPAR)

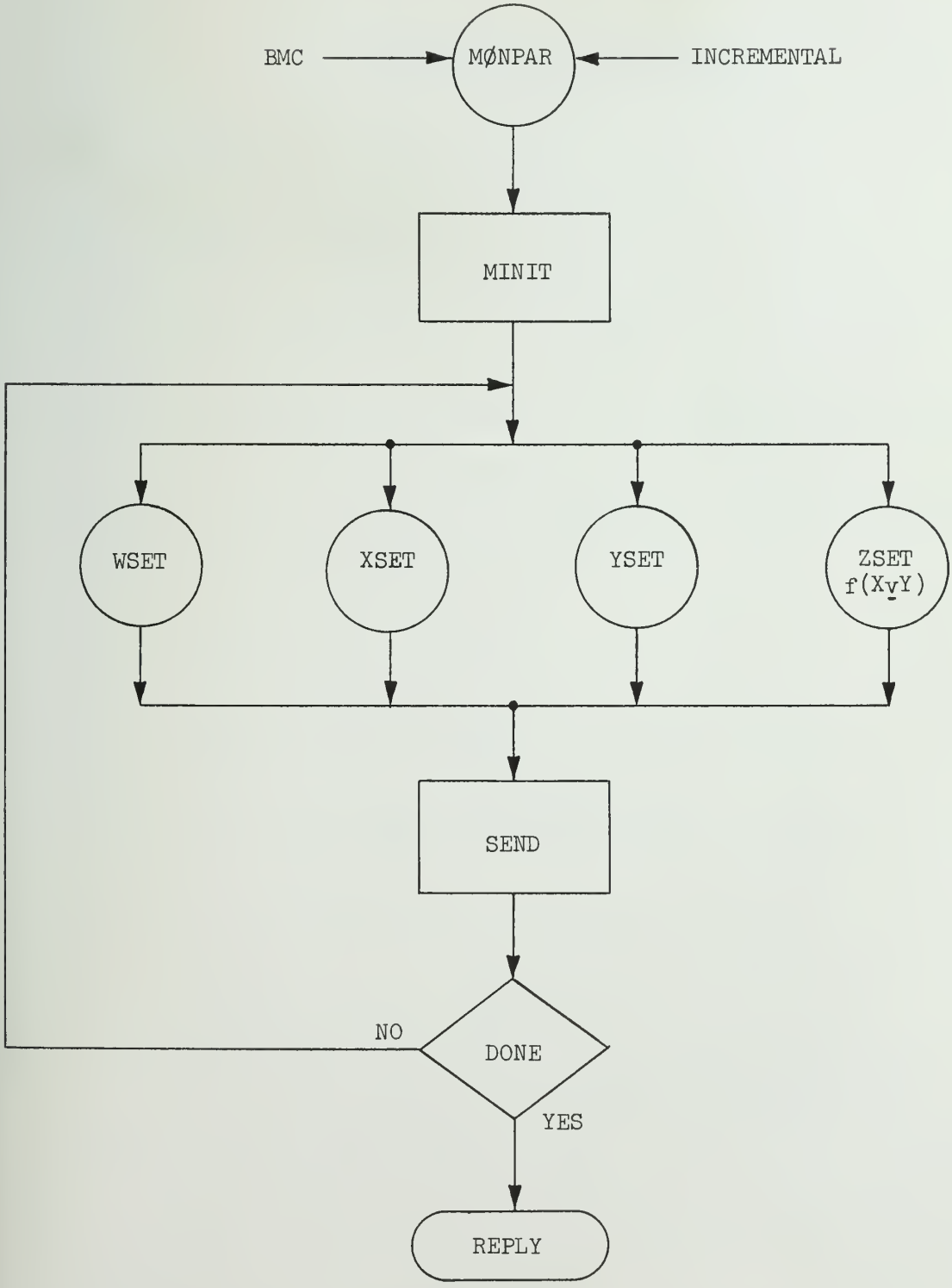


Figure 2.3.1 Flow Chart of MØNPAR

X-SET (Set x-axis parameters)

① If DPQX, go to ⑦

② If $\overline{A \cdot G} \cdot$(EXPX = p), go to ⑤

③ If $A \cdot G \cdot$(EXPX = q), go to ⑤

④ Go to ⑲

⑤ EXPX ← 0

⑥ DPQX ← 1

⑦ If DHX, go to ⑪

⑧ If $(\overline{\text{EXPX} = h})$, go to ⑲

⑨ EXPX ← 0

⑩ DHX ← 1

⑪ If $(\overline{\text{DF} = 2})$ + K, go to ⑭

⑫ If $A \cdot \overline{G}$, go to ⑮

⑬ If $(\overline{\text{EXPX} = n})$, go to ⑲

⑭ DN ← 1

⑮ MDCX ← 00

⑯ If $A \cdot \overline{G}$, go to ⑳

⑰ MDCZ ← 00

⑱ Go to ⑳

⑲ EXPX ← EXPX + 1

⑳ Reply

Table 2.3.3 Mathematical Designation for XSET

EXPX, 3 bit counter used to count up to values of p, q, h, and n

DPQX, (EXPX = p or q) flip-flop

DHX, (EXPX = h) flip-flop

DN, (EXPX = $\overline{n}$) flip-flop

MDCX, 2 bit register used for transfer of information to X control

MDCZ, 2 bit register used for transfer of information to Z control

A, scan-axis informational bit

G, rotation of scan-axis informational bit

p, 3 bits of x-axis incrementing step size information

q, 3 bits of y-axis incrementing step size information

h, 3 bits of magnification information

n, 2 bits of gray scale information

K, constant data or constant sample rate informational bit

(DF=2), raster type data format

Table 2.3.4 Identification of Terms for XSET

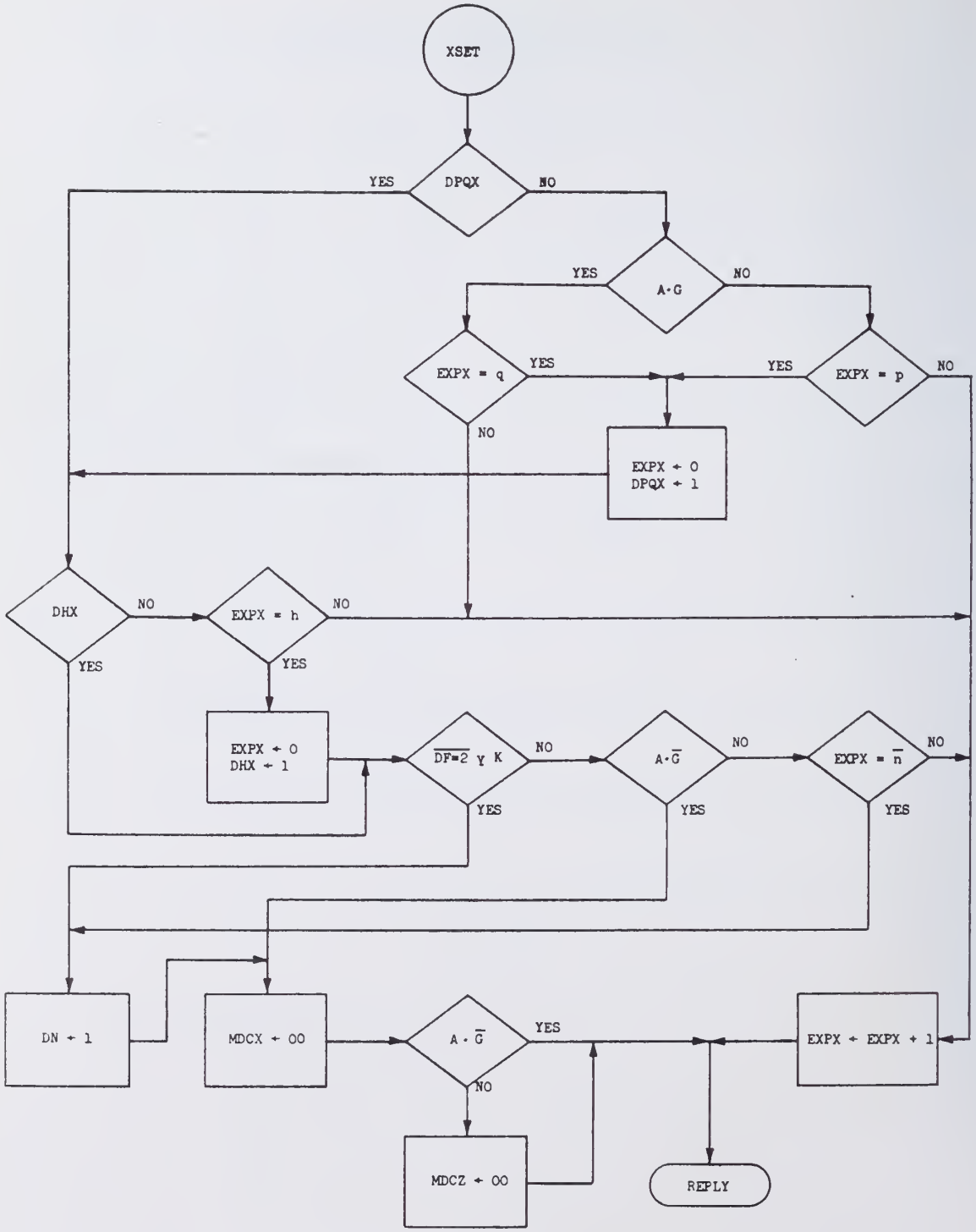


Figure 2.3.2 Flow Chart of XSET

3. CONTROL POINT - A BUILDING BLOCK APPROACH

3.1 Description

Pseudo-asynchronous control is based on the concept that tasks be performed in controlled steps or in irregular time intervals dependent on the execution time of these tasks. This type of control has been implemented at Illinois by the use of logical circuits called control point. The control point originally developed as a modification of the Illiac II "speed independent" control circuits by Gilles, Robertson, and Swartwout (1961). Generally speaking, each control step is performed by one control point. In many cases, the control point may be used to implement several similar control steps from different parts of a sequence and will return to that part of the sequence from which it was called.

The basic idea behind a control point is that it initiates some operation by turning on a given set of control lines. These control lines will remain on until either a certain length of time has passed, or a reply has been received indicating that the operation has been completed. As these control lines are turned off, an advance signal is initiated which will activate the next control point.

The use of control point, therefore, provides an ordered scheme for the assignment of specific tasks being performed in controlled steps. In other words, the control point renders to the logical designer a building block approach or technique in the design of pseudo-asynchronous control.

The evolution of the control point design for Illiac III has been long and tortuous. The investigation and applications of several control

point configurations have been made by Atkins and Nordmann (1969). The control point configuration presented in this paper consists of two stages: a task stage and a timing stage.

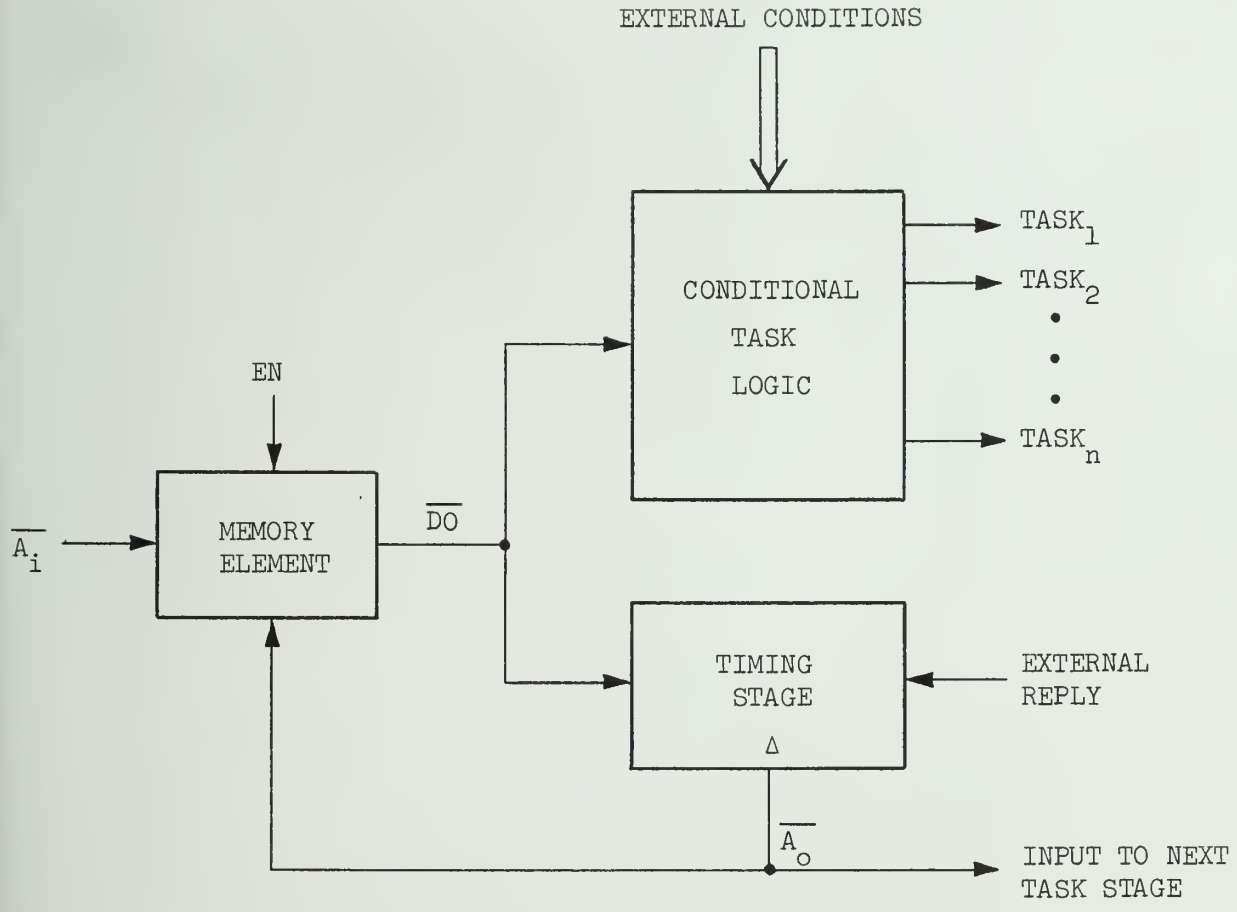
3.1.1 Task Stage

The function of the task stage is to perform certain operations on various hardware: i.e., gates are operated, flip-flops are set, counters incremented or other control points called. These operations may be conditional upon status conditions. Consider the block diagram of typical task stage logic as shown in Figure 3.1.1.1. The advance in line, $\overline{A}_i$, is connected to an adjacent stage logic. When this line drops to "0", the memory element is set and the task logic is said to be primed. When $\overline{A}_i$ returns to "1", the task activation signal, $\overline{DO}$, becomes "0" provided that the enable line, EN, is at "1". The task stage logic is now said to have been initiated.

The $\overline{DO}$ signal from the memory box is one input to the conditional task logic, while the other inputs to this logic are external conditions appropriate to that task stage. Typical examples of these conditions are the outputs of counter decoders, the contents of parameter registers, or the outputs of status flip-flops.

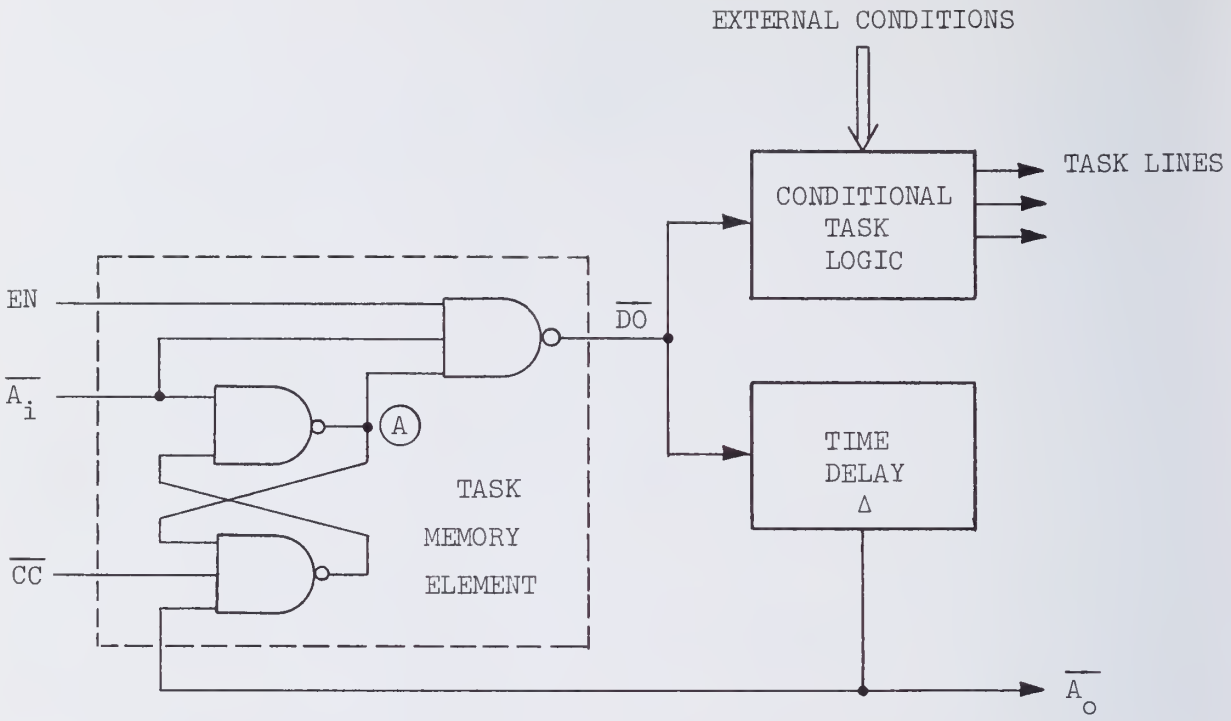
The $\overline{DO}$ signal also activates the timing stage which, after a selectable duration, causes the advance out line, $\overline{\overline{A}_o}$, to go to "0". This action will reset the memory element, thus turning off the task element, and can be used to also prime and initiate the next succeeding task stage.

Figure 3.1.1.2 illustrates the most elementary task stage configuration and an explanatory timing diagram. In this case the timing stage will consist of an internal timing model which will be explained later in this paper.

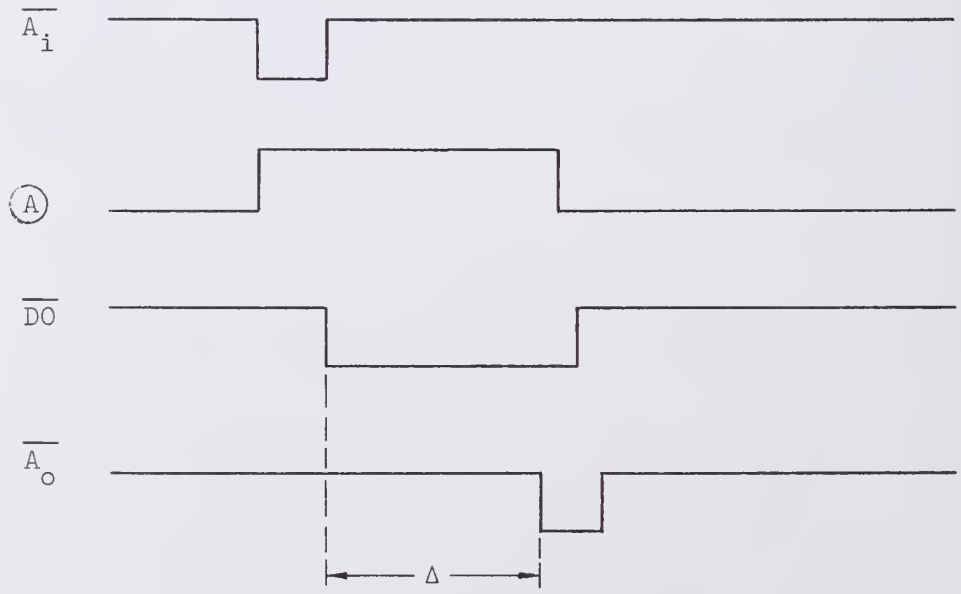


Δ = f (internal time delay or external reply signal)

Figure 3.1.1.1 Block Diagram of Task Stage Logic



TIMING DIAGRAM:



Assume that EN = 1 and $\overline{CC}$ = 1

Figure 3.1.1.2 Most Elementary Task Stage Configuration

No detailed logic is shown in the conditional task logic box since the configuration is highly variable from task stage to task stage. The most elementary configuration would be a direct connection of the $\overline{DO}$ (perhaps through an invertor or line driver) to the task lines.

The enable input,EN,offers a facility for inhibiting the operation of the task stage. If this input is held at "0", the control sequence will stop when the memory element of the inhibited stage is initiated. As EN returns to "1", the normal operation of $\overline{DO}$ will resume. This input may then be used to either inhibit a control sequence conditional upon an asynchronous signal or serve as a maintenance stop. A maintenance stop may be performed either manually as a function of a control panel switch, or automatically by the use of a diagnostic testing routine.

The common-clear input, $\overline{CC}$, provides a means by which the task stage memory element can be set to the "off" state. Typically the $\overline{CC}$ input of all the task stages of a routine(s) are connected together and "initialized" at the start of a specific control sequence.

It may be found desirable in some instances that a task stage be initiated from more than one advance in line. Figure 3.1.1.3 illustrates a method for implementing multiple advance in signals to a memory element. The $\overline{A}_i$ inputs are quiescently "1". When any one drops to "0", the memory element flip-flop is set such that Ⓐ = "1". As long as any $\overline{A}_i$ or the EN input is "0" the $\overline{DO}$ signal remains at "1", but when they return to "1" $\overline{DO}$ drops to activate the task logic. The duration of the $\overline{A}_{i_3} \cdots \overline{A}_{i_n}$ signals must be long enough to allow the memory flip-flops to set (i.e., compensate for the added propagation delay caused by the two additional NANDs).

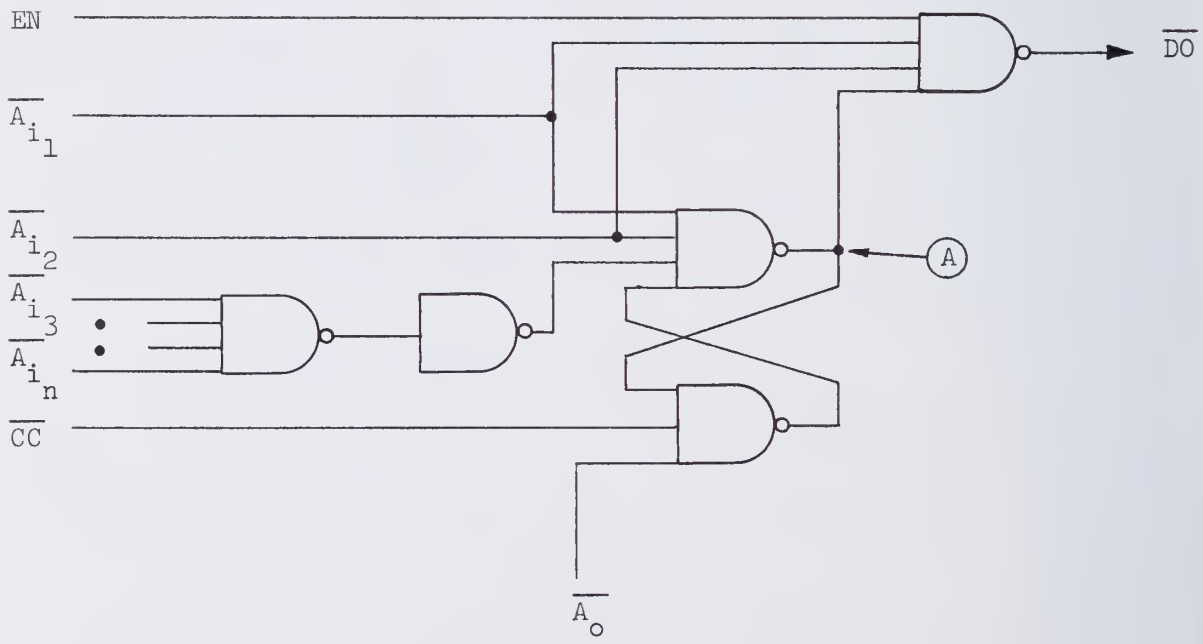


Figure 3.1.1.3 Multiple "Advance In" Inputs to Memory Element

3.1.2 Timing Stage

It is the function of this stage to provide a delay of the $\overline{DO}$ signal for a time period necessary to complete all the tasks of a given task stage. Associated with this time delay is the logic required for the generation of a reply signal $\overline{A}_o$ used to reset the task memory flip-flop, which turns off the task lines, and drives the next sequence stage logic which primes and initiates the next task stage.

In many cases this time delay is generated by an internal delay element which provides a timing model of the actual task. Figure 3.1.2.1 illustrates a timing stage configuration and an explanatory timing diagram. As $\overline{DO}$ goes to "0" (A), the output of NAND I, goes to "1". The advance out line, $\overline{A}_o$, will be delayed from going to "0" by the amount of time required for the RC-network at the (B) input of NAND II to charge to the logical "1" threshold. When this threshold is reached, $\overline{A}_o$ drops to "0" causing the task memory to be reset and therefore $\overline{DO}$ will return to "1".

The following design information on this circuit configuration was obtained by the use of basic circuit transformation techniques. A simplified model of NAND II and the equivalent circuits used in this analysis are shown in Figure 3.1.2.2. The logic elements used in this design were 54/74 series TTL and the following assumptions have been made.

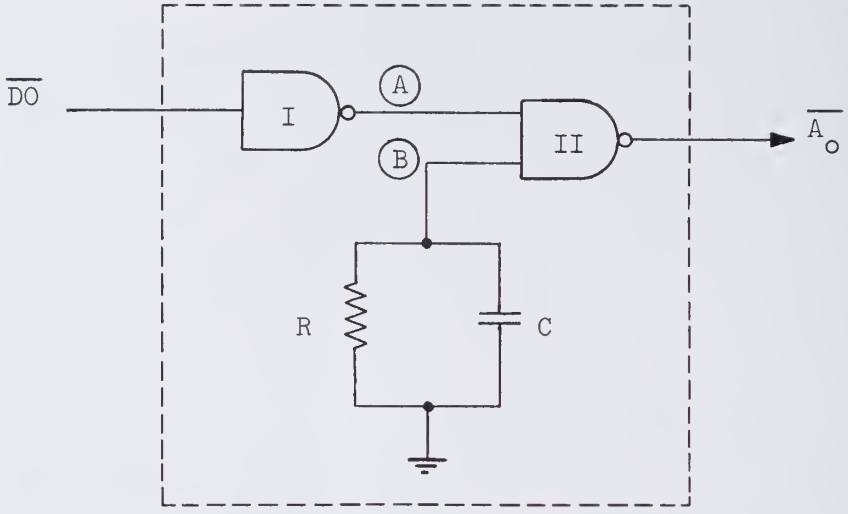
V_{cc} = 5.0 volts

$V_{THRESHOLD} = V_{(B)"1"}$ = 1.4 volts (midpoint of uncertainty range)

R_1 = 4 K Ω (value given in T.I.-TTL handbook)

R_2 = 8.2 K Ω (determined empirically)

C = 100 pfd.



TIMING DIAGRAM:

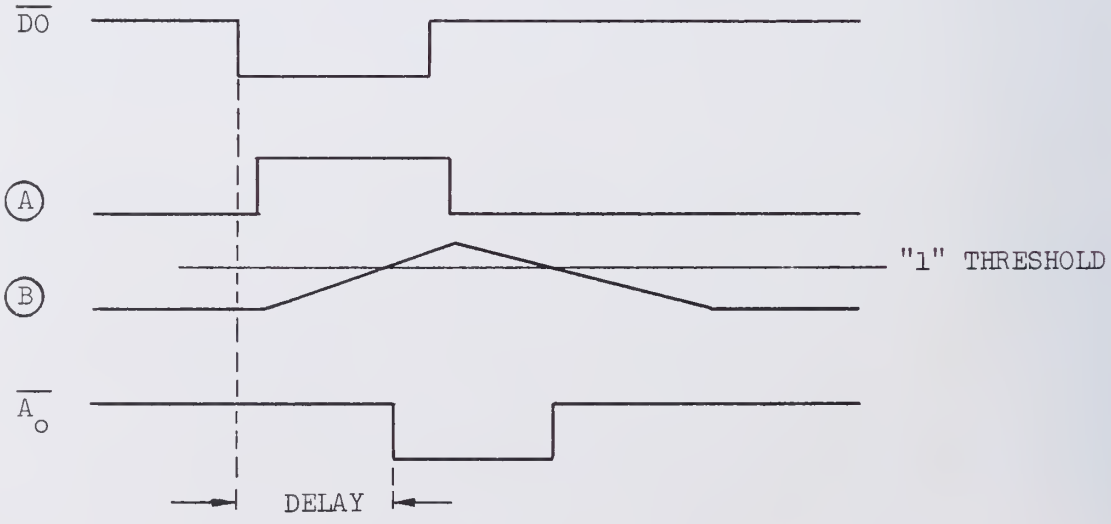


Figure 3.1.2.1 Timing Stage Using Internal Delay Element

Using the Thévenin equivalent circuit, the timing model has been reduced to a low-pass RC-network where

$$V_{TH} = V_{CC}\left(\frac{R_2}{R_1 + R_2}\right) = 5\left(\frac{8.2}{12.2}\right) = 3.36 \text{ volts}$$

$$R_{TH} = \frac{R_1 \ R_2}{R_1 + R_2} = 2.69 \text{ K } \Omega$$

The delay time is related then to the equation for the charging of capacitor C

$$V_C = V_{TH}\left(1 - e^{-t/(R_{th}C)}\right)$$

If we assume that V_C is initially zero and that the delay ends when V_C = threshold = 1.4 volts, then

$$1.4 = 3.36\left(1 - e^{-t/2.69 \times 10^{-7}}\right)$$

$$t = (.538 \times 2.69 \times 10^{-7}) = 145 \text{ nS}$$

The recovery time or the time required to discharge C is determined solely by R_2 and C. The discharge time can then be expressed by the equation:

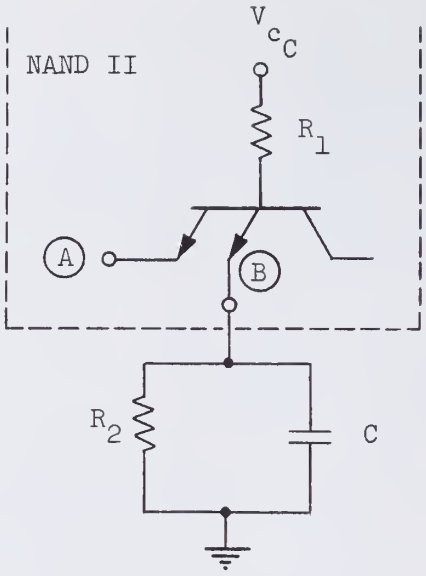
$$V_C = V_{TH}\left(e^{-t/R_2C}\right)$$

If we assume that V_C is initially equal to V_{TH} and that C is considered to be discharged when $V_C = .1V_{TH}$, then

$$.336 = 3.36\left(e^{-t/8.2 \times 10^{-7}}\right)$$

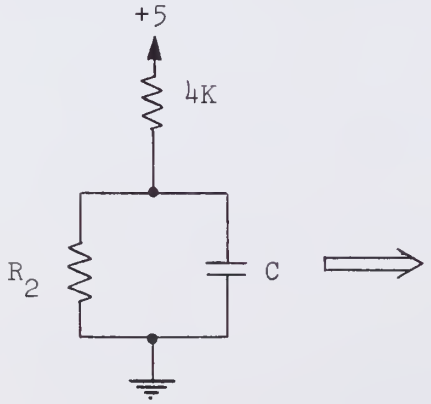
$$t = (2.3 \times 8.2 \times 10^{-7}) = 1885\text{nS}$$

It can then be seen from the above calculation that approximately 13 delay times must elapse between the application of input pulses at Ⓐ to

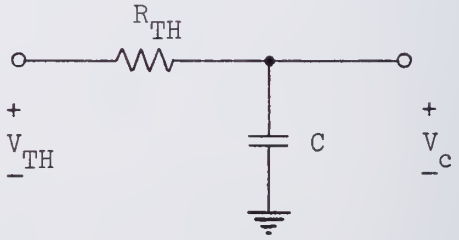


NAND II is typically SN7400N

EQUIVALENT CIRCUIT



THEVENIN'S EQUIVALENT



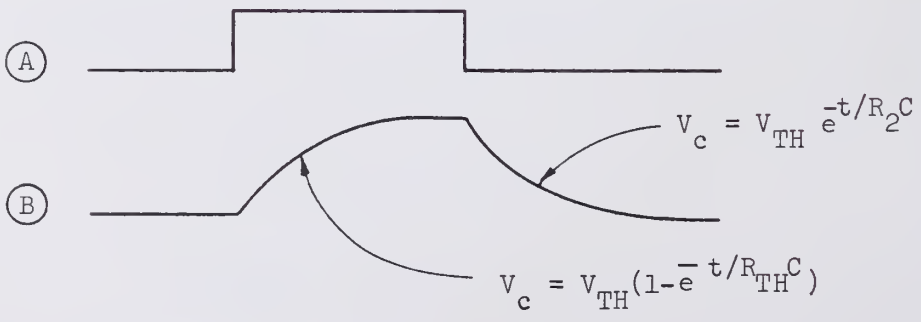


Figure 3.1.2.2 Timing Stage Equivalent Circuits

avoid interaction. If this constraint hinders circuit performance it may be greatly reduced by the addition of a diode as shown in Figure 3.1.2.3. This diode, a USD25 is a HP2800 hot carrier device with a low forward drop and junction capacitance, is used to discharge C when (A) goes to "0". With the addition of this diode, only 1.5 delay times are required between input pulses or the timing stage can be pulsed as soon as it has completed its delay function.

Empirical tests have indicated a design center value for R_2 of 8.2KΩ with upper and lower limits of 12K and 6.2K respectively. Using R_2 = 8.2KΩ, the following was found to hold:

Delay - 1.5nS/pFd.

Variation of $\pm$ .5v in V_{CC} - $\pm$ 15%

Variation between IC packages - $\pm$ 15%

Variation due to R_2 = 6.2K → 12K - 8%

The delay time of the timing stage could also be a direct function of an external replay signal as has been shown in Figure 3.1.2.4. In this case the RC-network has been replaced by the reply line, GO.

Once again as in the previous example, as $\overline{DO}$ goes to "0", the output of NAND I goes to "1". The advance out line $\overline{A}_o$ will not go to "0" until GO has been set to "1". If GO were "1" as $\overline{DO}$ goes to "0", the delay time would then depend on the propagation time of NAND I and NAND II and the amount of time required for the task flip-flop to be reset by $\overline{A}_o$.

The control point is then the combination of a task stage and a timing stage with the block diagram and circuit configuration being represented in Figure 3.1.2.5. and actual test results are presented in Appendix II.

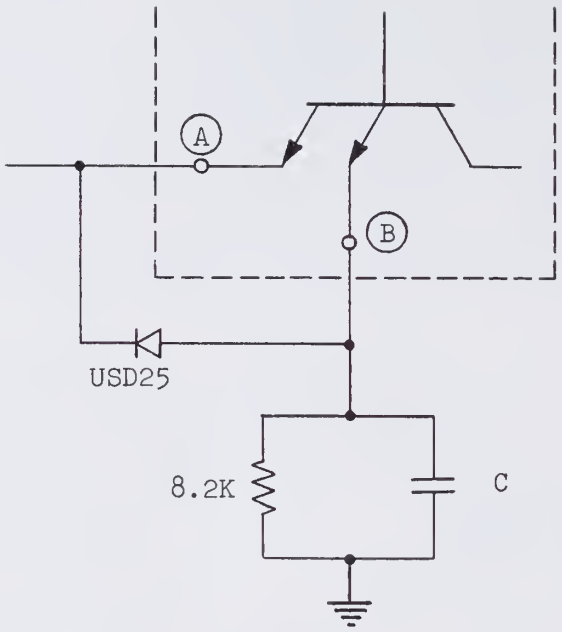


Figure 3.1.2.3 Delay Circuit with Diode to Enhance Recovery

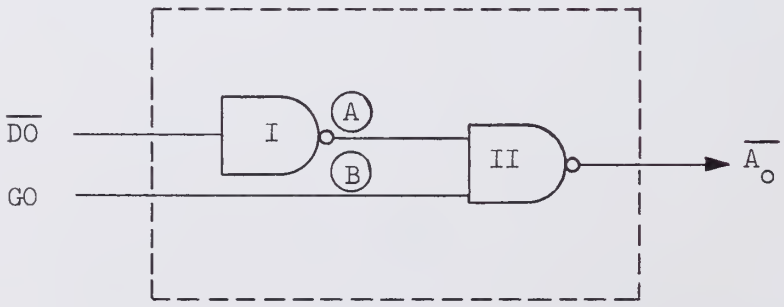
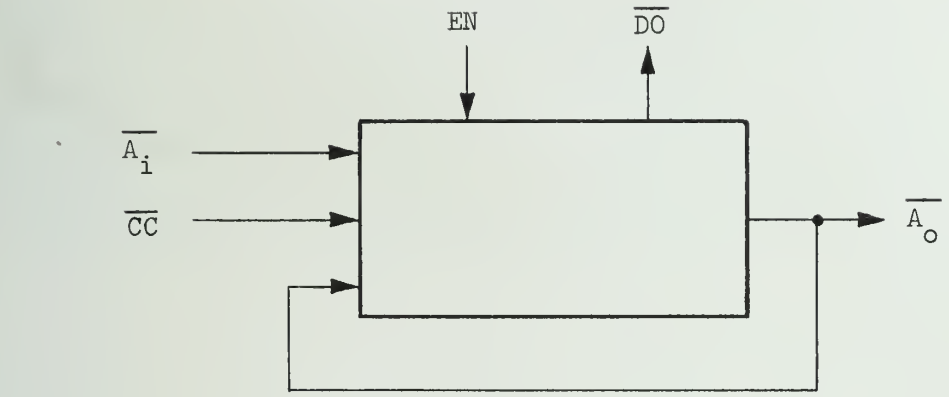
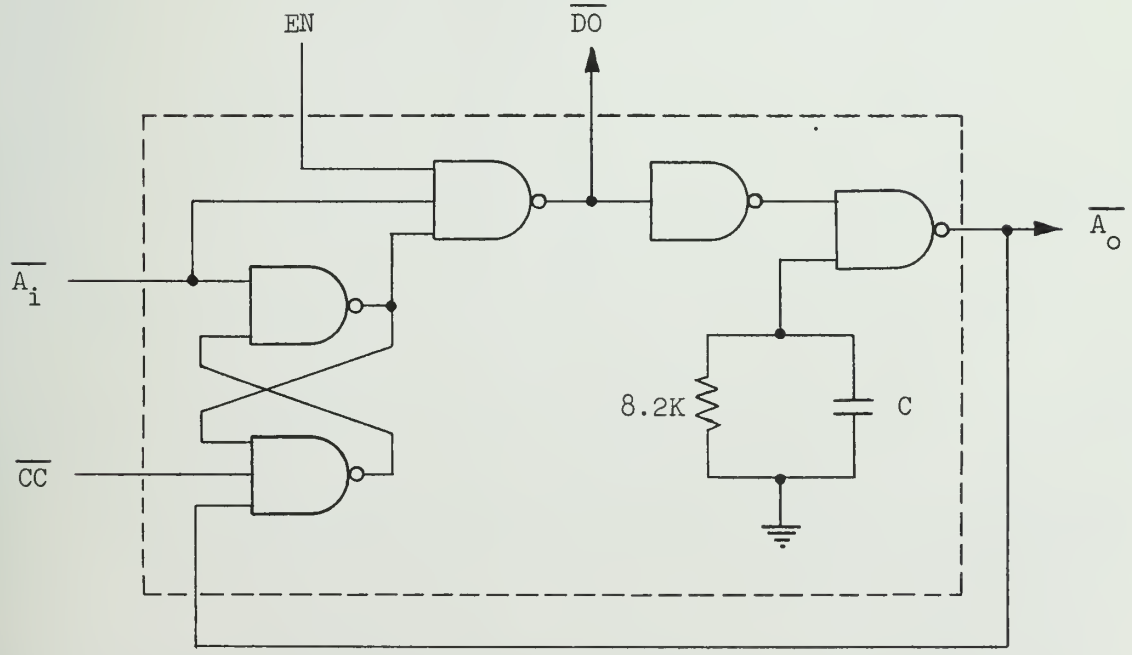


Figure 3.1.2.4 Timing Stage Using External Reply





where C = f (time delay required)

Figure 3.1.2.5 Control Point Block Diagram and Circuit Configuration

3.2 Sequence Stage

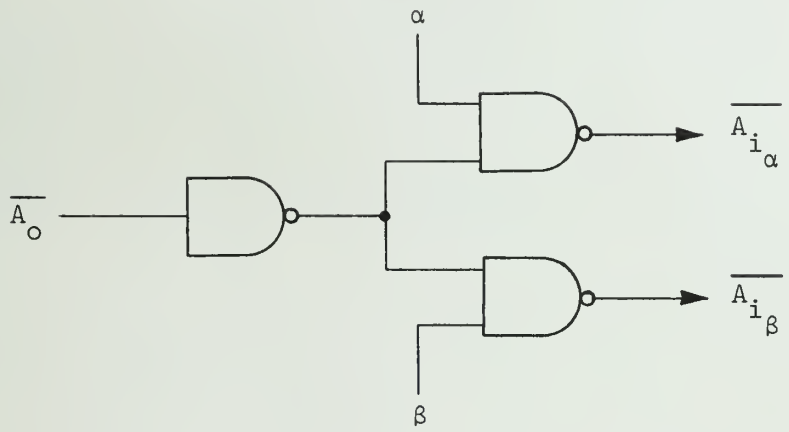
Interspersed with the control points used to implement the asynchronous controlled steps is the sequence stage. The function of the stage is basically one of selection or steering. That is, as a control point completes its specific task the sequence stage makes the decision as to which control point(s) to initiate next.

The most elementary example of sequence stage logic is merely a wire connecting the $\overline{A}_o$ output of one control point to the $\overline{A}_i$ input of the next control point.

In the case where two or more control points are to be called concurrently, the sequence stage logic will be AND'ed as a function of $\overline{A}_o$ and external conditions. Figure 3.2.1 illustrates an example of a two-way branch. The conditions, α and β, determine where the $\overline{A}_o$ pulse from the previous control point is directed. It is important that these conditions are set prior to the arrival of the $\overline{A}_o$ pulse, therefore, it is advisable that α or β not be determined by the action of the control point which generates the $\overline{A}_o$ pulse which they steer. It should be noted that at least one condition must be true or else the $\overline{A}_o$ pulse is lost and the control sequence hangs-up i.e., if $\alpha = \beta = 0$, an error exists.

In many cases the conditional logic is designed such that $\alpha = \overline{\beta}$ and this error situation is averted.

The sequence stage may also be required to delay the continuation of control contingent upon an asynchronous wait condition. Since the arrival time of this signal is unknown, the circuit configuration of Figure 3.2.1 is not applicable. Figure 3.2.2 and Figure 3.2.3 illustrates methods of implementing this wait condition.



α, β ARE BRANCHING CONDITIONS

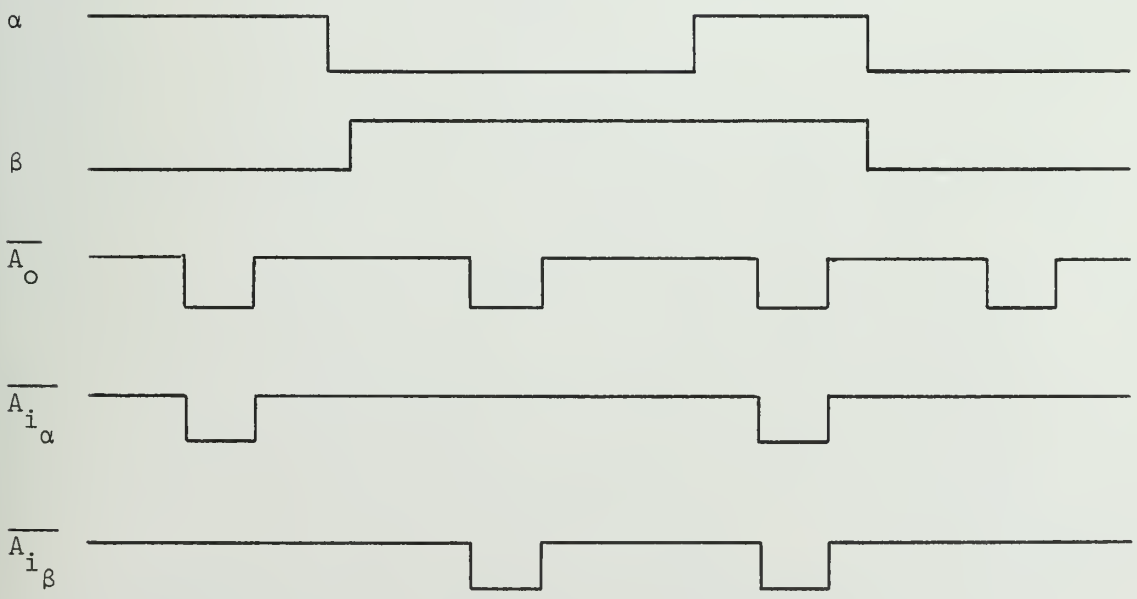


Figure 3.2.1 Two Way Branch Sequence Stage Logic

In Figure 3.2.2 the wait signal is used to delay the action of control point after it has been initiated. That is, $\overline{DO}$ will be inhibited from going to "0" until WAIT goes to "0". When the wait condition is satisfied, $\overline{DO}$ provides a means for keeping EN at "1" until the control point completes its normal operation. This same delay action to the control point operation could also be a function of a maintenance halt, MH, used for check-out and diagnostic procedures.

In Figure 3.2.3 the wait signal is used to delay the propagation of the $\overline{A}_o$' signal. The $\overline{DO}$ signal is then a function of both the timing stage and the wait condition. Here the $\overline{DO}$ line will drop to "0" and remain there (even though the timing stage is done) until WAIT goes to "0". The $\overline{A}_o$' signal will then terminate the operation of the control point and initiate the next stage logic.

This same wait logic configuration may also be used to interlock two or more parallel, independent control chains. The design requirement here is to make the $\overline{A}_o$ signal to the next stage wait until all the parallel tasks are complete. Figure 3.2.4 illustrates an example of this interlocked control chains. The $\overline{A}_o$ line of last control point of each chain is delay until a reply from all the chains is received. It is assumed that when one control chain is activated, then so is the other, i.e., that eventually both replies will be generated.

Another application of the wait logic, similar to that shown in Figure 3.2.3, is where a control point is used to call some routine and will not advance control until the called routine has replied. For apparent reasons, this configuration has been named "calling control point" and is

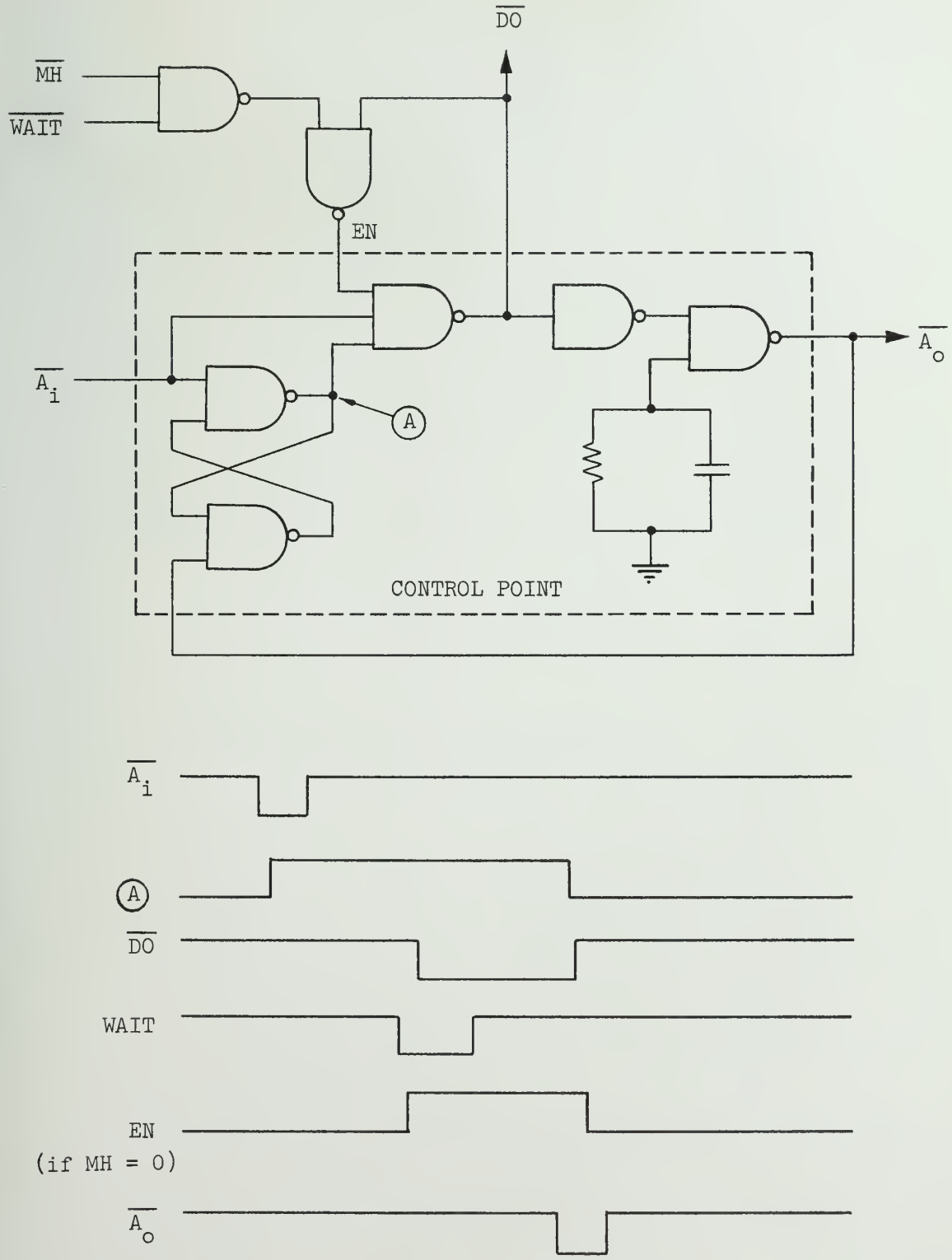
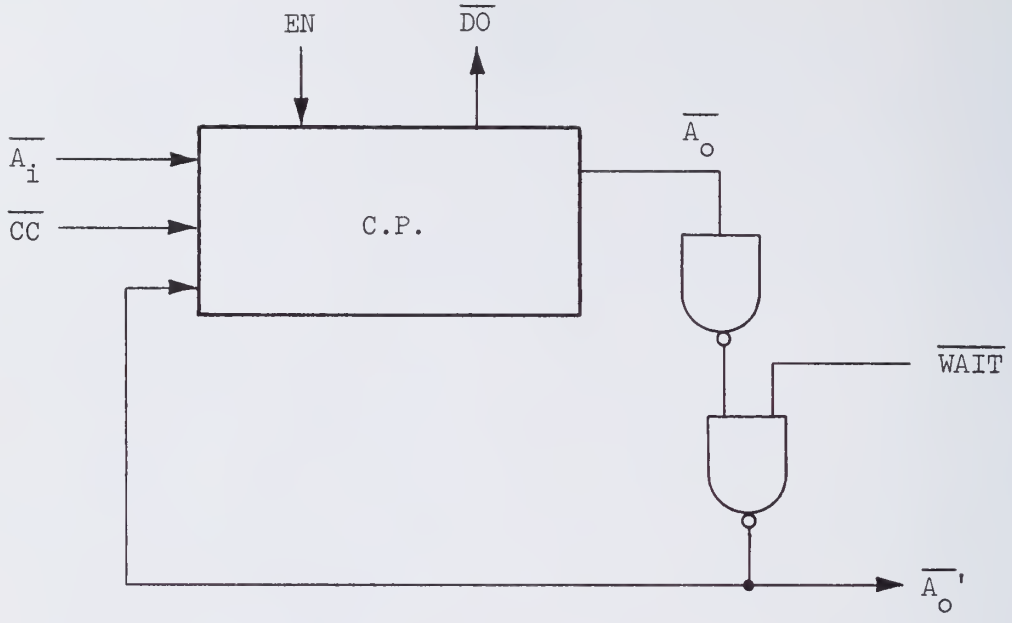


Figure 3.2.2 WAIT Condition Using EN Gate



ASSUME THAT EN = $\overline{CC}$ = 1

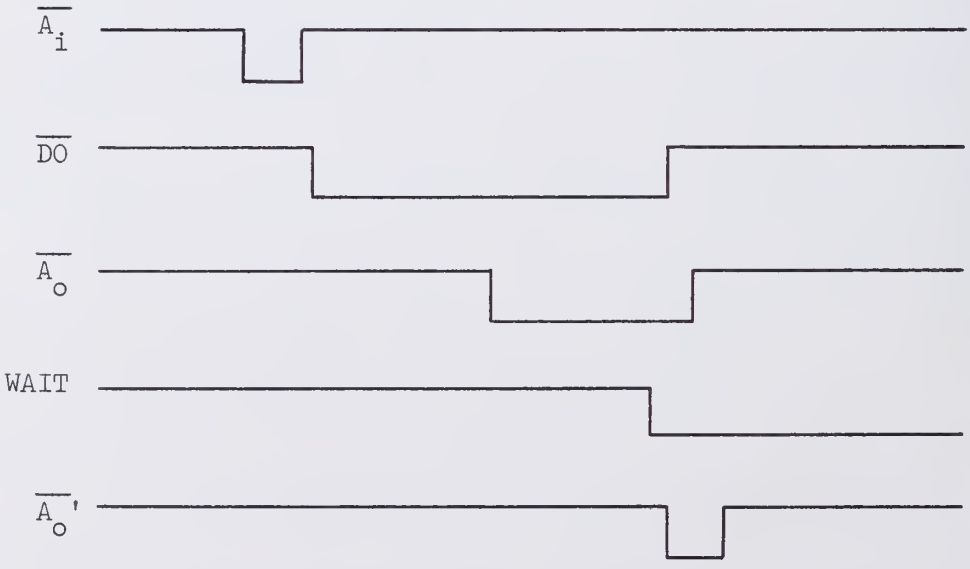
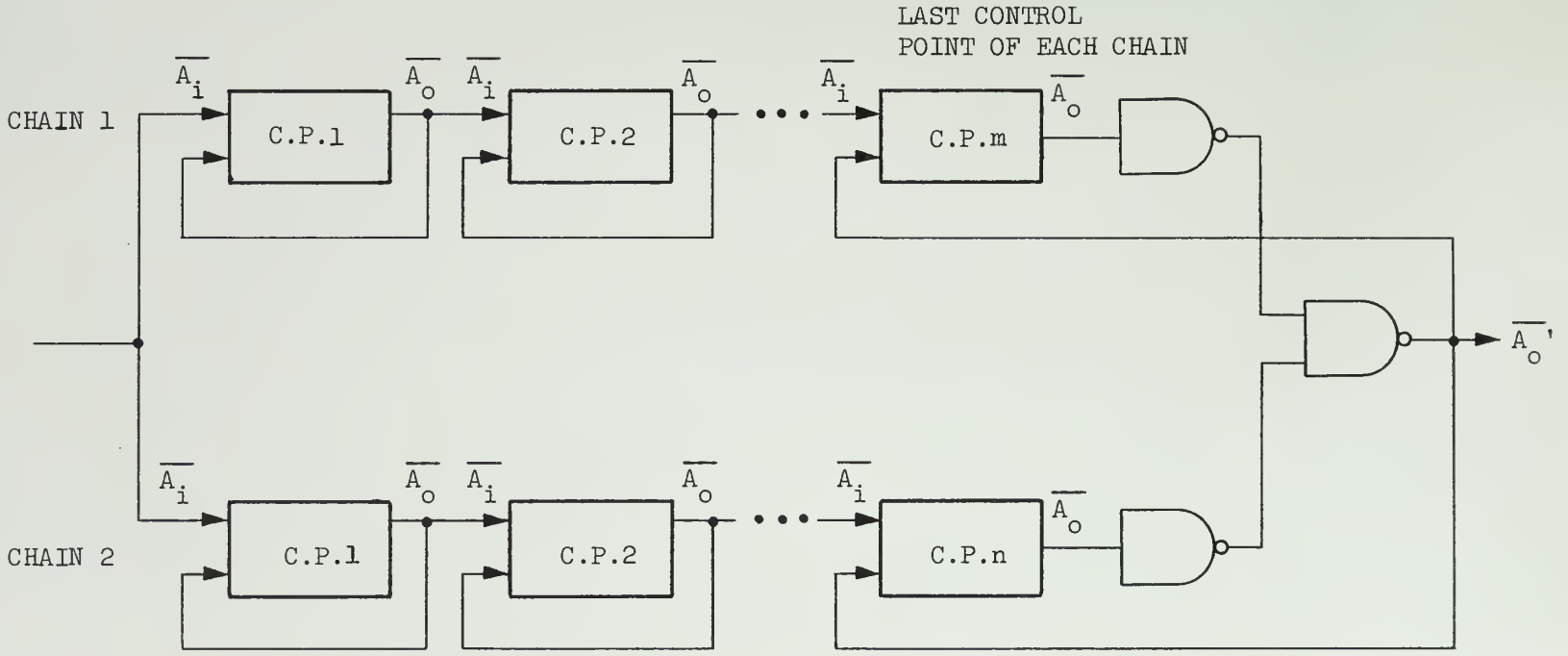


Figure 3.2.3 Wait Condition Using $\overline{A_o}$ Line

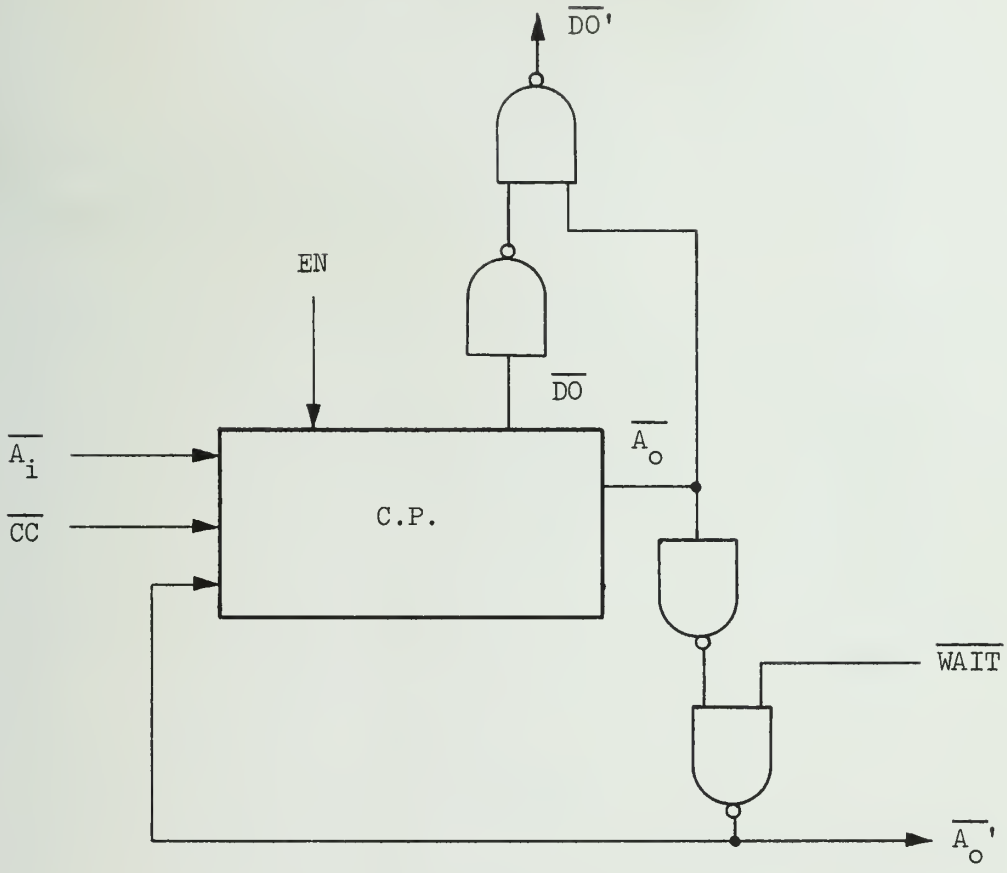


NOTE THAT EACH CHAIN NOT NECESSARILY THE SAME LENGTH

i.e. $m \neq n$

Figure 3.2.4 Interlocking Two Parallel Control Chains

shown in Figure 3.2.5. The $\overline{DO}'$ line will go to "0" as the control point is initiated (i.e., $\overline{DO}$ goes to "0" and $\overline{A}_O$ is at "1"). After a specific delay time, due to the timing stage, $\overline{A}_O$ goes to "0" causing $\overline{DO}'$ to return to "1". This action of $\overline{DO}'$ may have been used to prime and initiate a control point in a routine. When this routine has completed its task, it replies and WAIT goes to "0".



ASSUME THAT EN = $\overline{CC}$ = 1

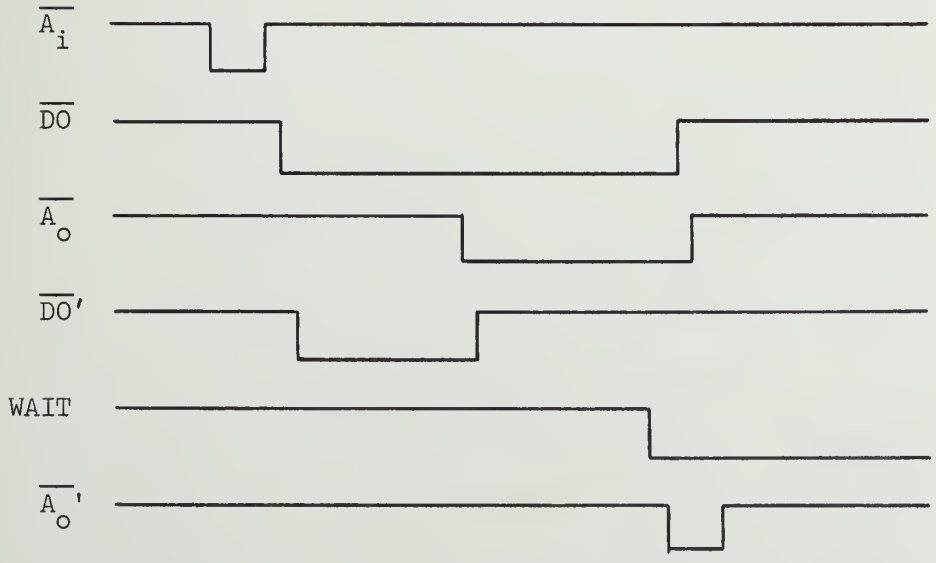


Figure 3.2.5 "Calling Control Point" Circuit Configuration

4. CONTROL LOGIC IMPLEMENTATION

4.1 Conversion of Flow Charts to Control Logic Representation

The purpose of this section is to describe the procedure used in the realization of control design by using techniques previously detailed in Sections 2 and 3.

In general, it is assumed that the basic control design initially exists in the form of flow charts. It is important that these flow charts be representative of the techniques employed in Section 2. That is, the flow chart should be in a very explicit form so that every control signal which must be influenced is expressed. Therefore, each operation(s) which relates to each given control step should be written as the contents of the operational block or decision symbol of the flow chart. Each control step thus becomes in effect a list of either the control signals to be activated by the task signal of a given operational block (control point) or the specification of a decision to be performed (sequence stage).

Careful inspection of the flow charts may reveal that many single control steps or even whole sequences of control steps are identical. Such occurrences are often capable of being implemented by the same physical control point(s) and should be noted for future reference.

After the flow charts have been carefully scrutinized and are considered as being optimized and representative of the prescribed format, the next step in design implementation is that of converting the flow charts into actual logic drawings. In general, certain conventions will be employed for making these conversions and are shown in a chart in Table 4.1.1. It is rather apparent from this chart that the majority of flow chart symbols are represented by control points in the logical drawings. Additional logic

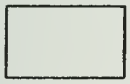
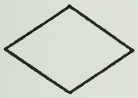
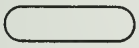
FLOW CHART		LOGIC DRAWING	
SYMBOL	NAME	CIRCUIT REPRESENTATION	ADDITIONAL LOGIC
	ROUTINE	CONTROL POINT	DEPENDS ON RELATIVE POSITION IN FLOW CHART
	TASK	CONTROL POINT	CONDITIONAL LOGIC
	DECISION	SEQUENCE STAGE	———
	REPLY	CONTROL POINT	REPLY FLIP-FLOP

Table 4.1.1 Conversion of Flow Chart to Logic Representation

elements may also be required to fully realize these conversions. In many instances this logic will be incorporated as part of the sequence stage.

The ROUTINE symbol may appear as either the first entry of a flow chart or as a control step within the chart. This symbol can then be logically represented by a control point plus additional logic dependent upon the relative position of the symbol in the chart.

In the first case the function of this symbol is to: 1) indicate the entry into a specifically named routine, 2) list the sources for the activation of this routine, 3) reset the reply flip-flop and set the conditional flip-flops used in this routine. A single control point is all that is needed to implement this function.

In the second case the function of the ROUTINE symbol is to: 1) call another routine, 2) provide a return of control to the original routine upon receiving a completion indication from the called routine. The operation here may be implemented by using a control point and the wait logic as shown in Figure 3.2.5.

The implementation of the TASK symbol is achieved by using a control point and the various additional gates needed for the conditional task logic. This task logic is highly variable and dependent upon the conditions associated with each task stage. It is the responsibility of the designer then to specify the logic elements required for each task stage.

The DECISION symbol is normally realized by using either the sequence stage as shown in Figure 3.2.1 or the wait logic as shown in Figure 3.2.3.

The implementation of the REPLY symbol is accomplished by a control point and a reply flip-flop which can be used as a routine or sequence status indication.

The symbols of the flow chart are connected together by lines which in most cases are representative of the sequence stage. The general configurations for sequence stages, as explained in Section 3.2, should be employed here.

In making logic drawings for a large sequence or control it is usually easier simply to label the task signals and then on a separate drawing show all the collection logic and signal drivers associated with these task signals. This is extremely helpful if a given task may be signaled from many separate routines. The collection logic is usually combinations of NOR's and NAND's which fan into the signal or line drive which then drives the actual control signal. The outermost layer of this logic will consist of NAND's and NOR's whose inputs are the labeled task signals of the various routines or sequences.

Optimization of this logic can be accomplished by looking for certain types of tasks which always use the same sets of control lines.

Often several redrawings of the control point logic will be necessary in order to optimize the logic between control points and the conditional task logic.

The application of these conversion techniques has been illustrated in Figure 4.1.1 for the MØNPAR sequence which was described in section 2.3.

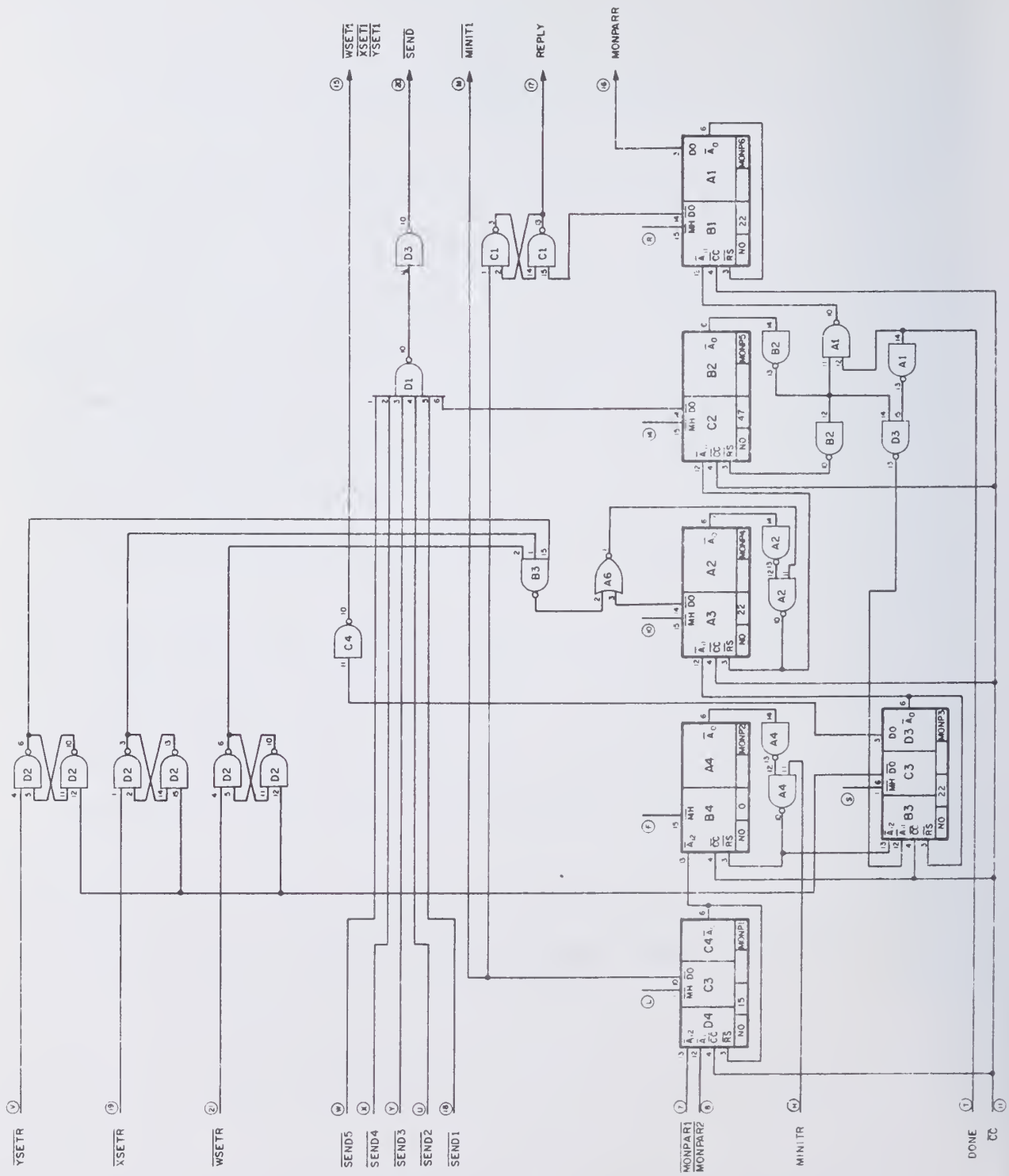


Figure 4.1.1 Logic Drawing for MØNPAR

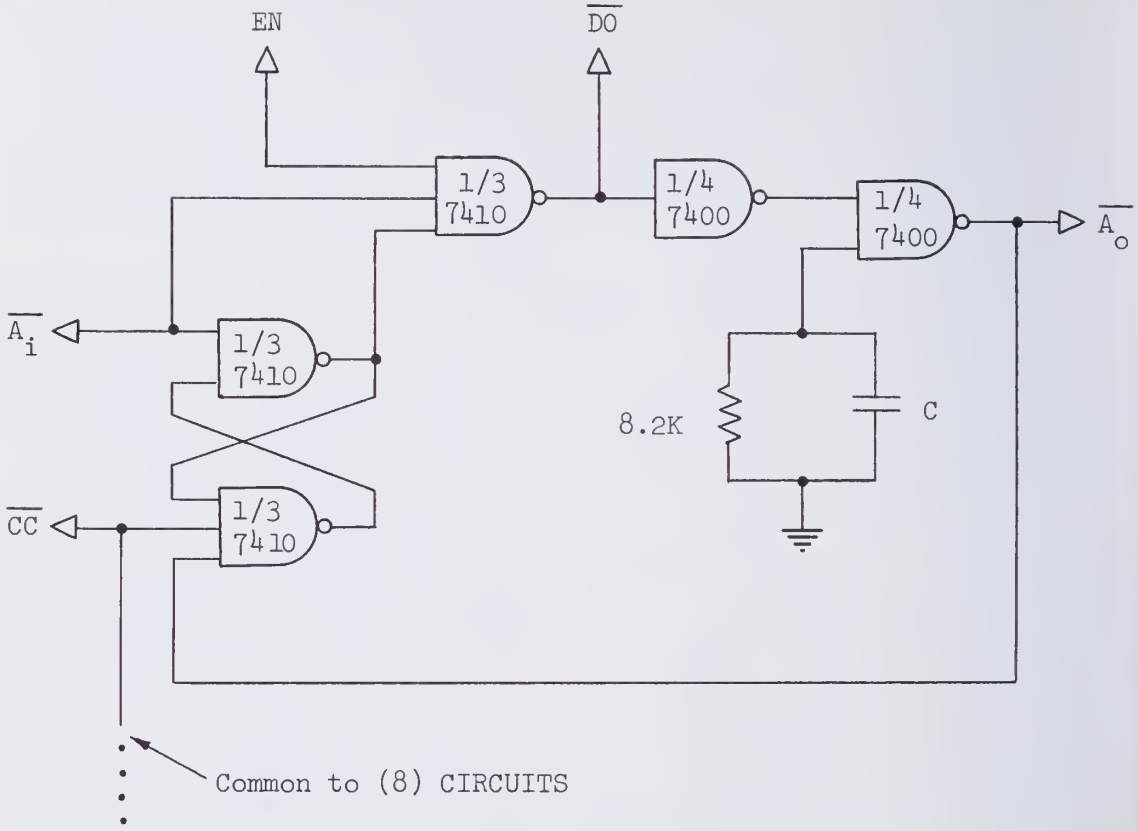
4.2 Hardware Realization

After the control design has been implemented in terms of logical drawings, the problem exists as to how might this design be realized in actual logic elements. Many factors such as cost, speed, packaging, operating parameters, and reliability must be evaluated in the selection of these logic elements.

The control logic of Illiac III has been implemented with Texas Instruments 74N Series TTL logic. Several etched board configurations have been designated to facilitate the application of these logic elements.

One board consists of just control points (i.e. task and timing stages) and is illustrated in Figure 4.2.1. Any sequence stage logic, task condition logic and any other reply generation logic other than a single delay element will be implemented on a universal IC board. This board provides facilities for 24, dual inline IC packages (14 or 16 pin) and includes 44 card edge pins which mate with the back-panel wired connectors of the mainframe.

In many instances it is advantageous to use the universal board to implement the logic for a whole sequence or routine. If this is the case, the control points are also included on this same board. Each card is then unique in that it has been wired (wires are connected between IC socket pins on the back of the card) to perform a certain function or control routine. This tends to give the control a modular appearance and reduces the amount of wiring normally required to interconnect a control point card with a conditional logic card. This also facilitates the check-out or bench testing capabilities of a given routine. Examples of these hardware realization techniques are shown in the logic drawings of Appendix I. It should be noted that the



NOTE: 1. ▷ INDICATES EXTERNAL PIN
2. (8) CONFIGURATION PER BOARD
3. VALUE OF C DEPENDENT ON FUNCTION OF $\overline{DO}$

Figure 4.2.1 Control Point Card

EN gate of all control points are connected to the etched board output pins to provide maintenance capabilities.

This provides the capability for "stepping" the control sequence or routine through its normal operating functions at a level of individual control point increments. This has proven extremely useful in control check-out and the detection of erroneous task operation.

There still remains the specification of delay time required for the control point to perform its task. In some cases the control point may function in a speed-independent manner and therefore no delay capacitor is used in the timing stage. However if a delay capacitor is to be used, it is the responsibility of the logic designer to select a value of capacitance that will guarantee that the control point properly executes its task(s). Some design information related to control point operation is available in Appendix II.

4.3 Question of Minimization vs. Standardization

While the main theme of this paper has been concerned with the standardization of control design implementation, the hardware realization may not necessarily be of an optimized nature. That is, it is possible that not all the logic elements of a given IC chip have been used. It is also possible that not all the available IC socket locations are utilized due to the uniqueness of the design function or the output pin limitation of the etched board.

In many cases these constraints have proven to be advantageous instead of being a disadvantage. It is often times necessary to make modifications and changes to the logic design as determined by the results of control check-out and testing. An extra logic element or IC socket location can often times be very handy in making these modifications. It is then the

responsibility of the logic designer to specify the type of logic elements (NAND's, NOR's, latches, drivers, etc.) to be used in a standardized control implementation approach while still trying to maintain a minimization of logic elements.

CONCLUSION

Since the design of a pseudo-asynchronous control logic does not necessarily produce a unique topology, a design technique or method has been developed to provide a standardized design approach. The specific technique chosen produced a comprehensive set of sequentially ordered control diagrams (flow charts) which could be converted directly to logical drawings using certain standard modeling procedures (control point). Additional consideration was given to such areas as control reliability, hardware realization, and the ease of control maintenance.

The techniques and procedures offered in this paper were used in the design and implementation of the Illiac III Scanner-Monitor-Video Controller and proved to be extremely helpful in the realization of this control.

While these techniques were applied to the control design for a specific computer, they may be applicable to other design disciplines. In particular, there appears to be a correlation to the work of Clark (1966) on the macromodular approach to computer design.

LIST OF REFERENCES

Atkins, D. E., (December 1969), "Illiac III Computer System Manual: Arithmetic Units", Vol. I, Illinois Department of Computer Science Report No. 366

Braun, E. L., (March 1963), "Digital Computer Design", Academic Press, New York

Clark, W. A., (February, 1966), "A Macromodular Approach To Computer Design", Washington University Computer Research Laboratory Technical Report No. 1

Friedman, A. D., (June 1968), "Synthesis of Asynchronous Sequential Circuits with Multiple Input Changes", IEEE Transactions on Computers, Vol. C-17, No. 6

Gillies, D. B., (August 1961), "A Flow Chart Notation for the Description of a Speed Independent Control", Department of Computer Science File No. 386

Langdon, G. G., (December 1968), "Analysis of Asynchronous Circuits Under Different Delay Assumptions", IEEE Transactions on Computers, Vol. C-17, No. 12

Nordmann, B. J., (September 1969), "Illiac III Computer System Manual: Taxicrinic Processor", Department of Computer Science Report (In process)

Robertson, J. E., (August 1961), "Problems in the Physical Realization of Speed-Independent Control", Department of Computer Science File No. 387

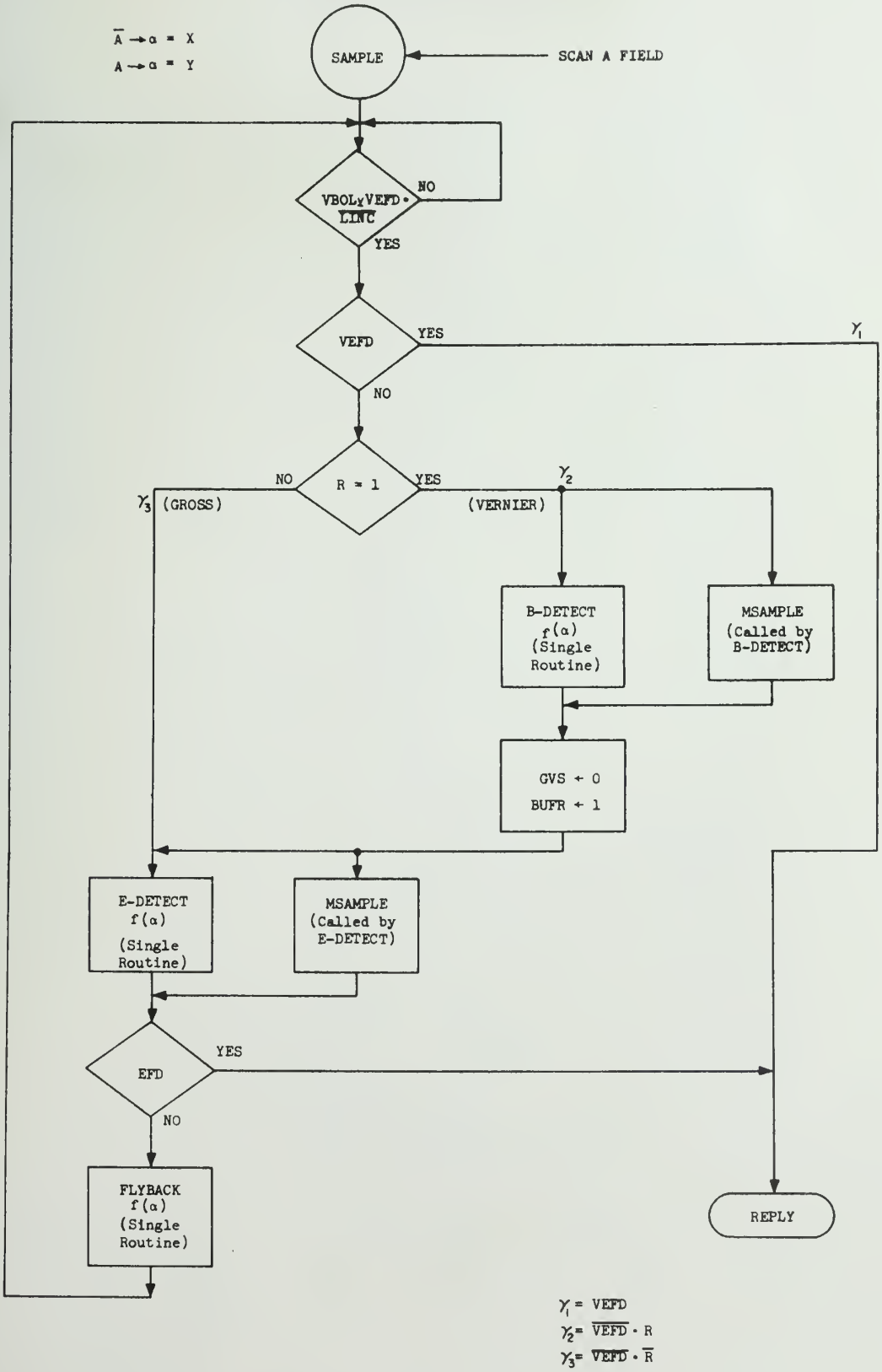
Swartwout, R. E., (August 1961), "One Method for Designing Speed-Independent Logic for a Control", Department of Computer Science File No. 388

Swartwout, R. E., (December 1963), "Further Studies in Speed-Independent Logic for a Control", PhD. Thesis, University of Illinois, Department of Computer Science Report No. 130

Appendix I

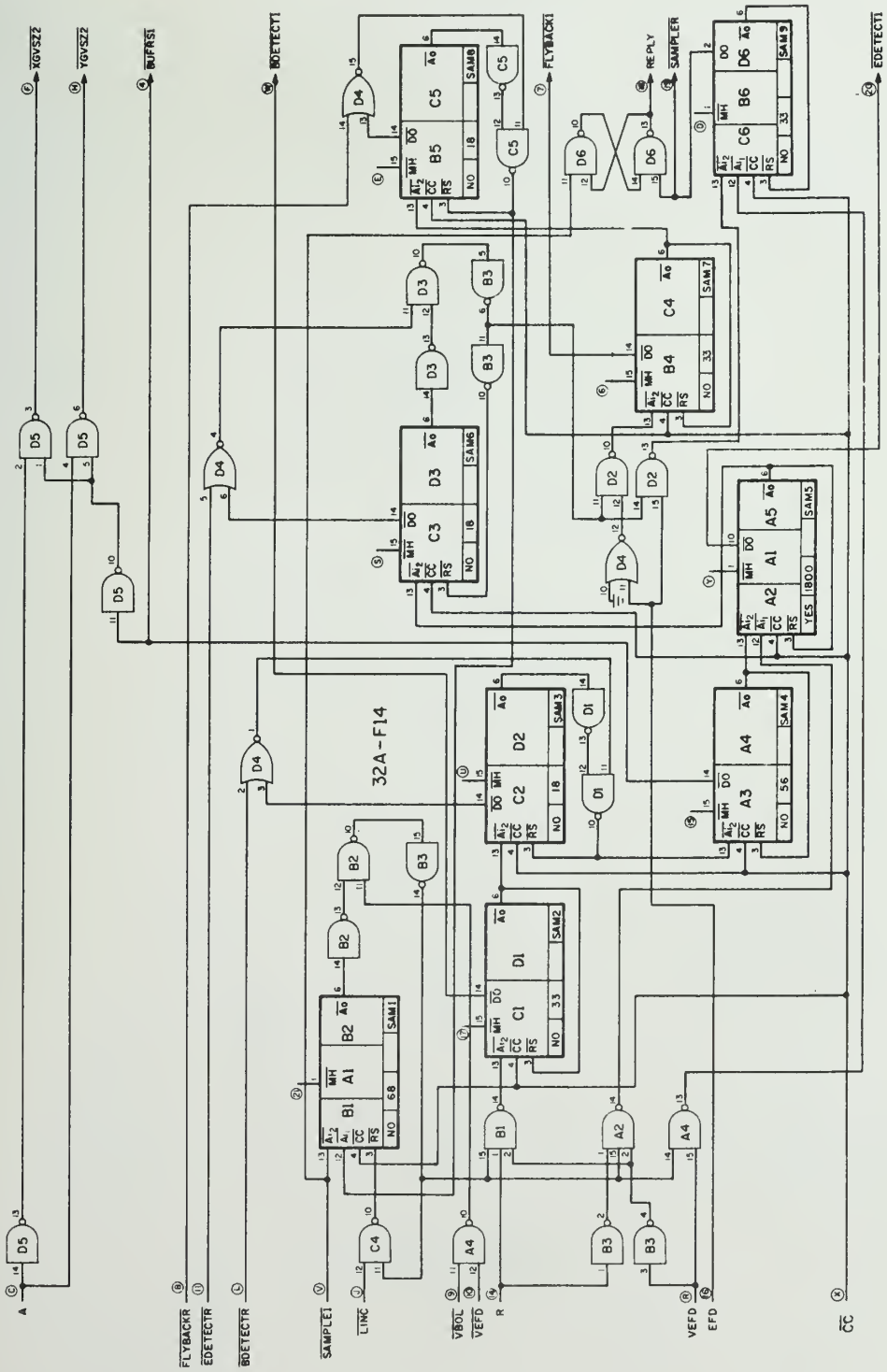
This appendix contains actual flow charts and their logical representations for control routines of the Scanner-Monitor-Video Controller of Illiac III.

Flow Chart of SAMPLE

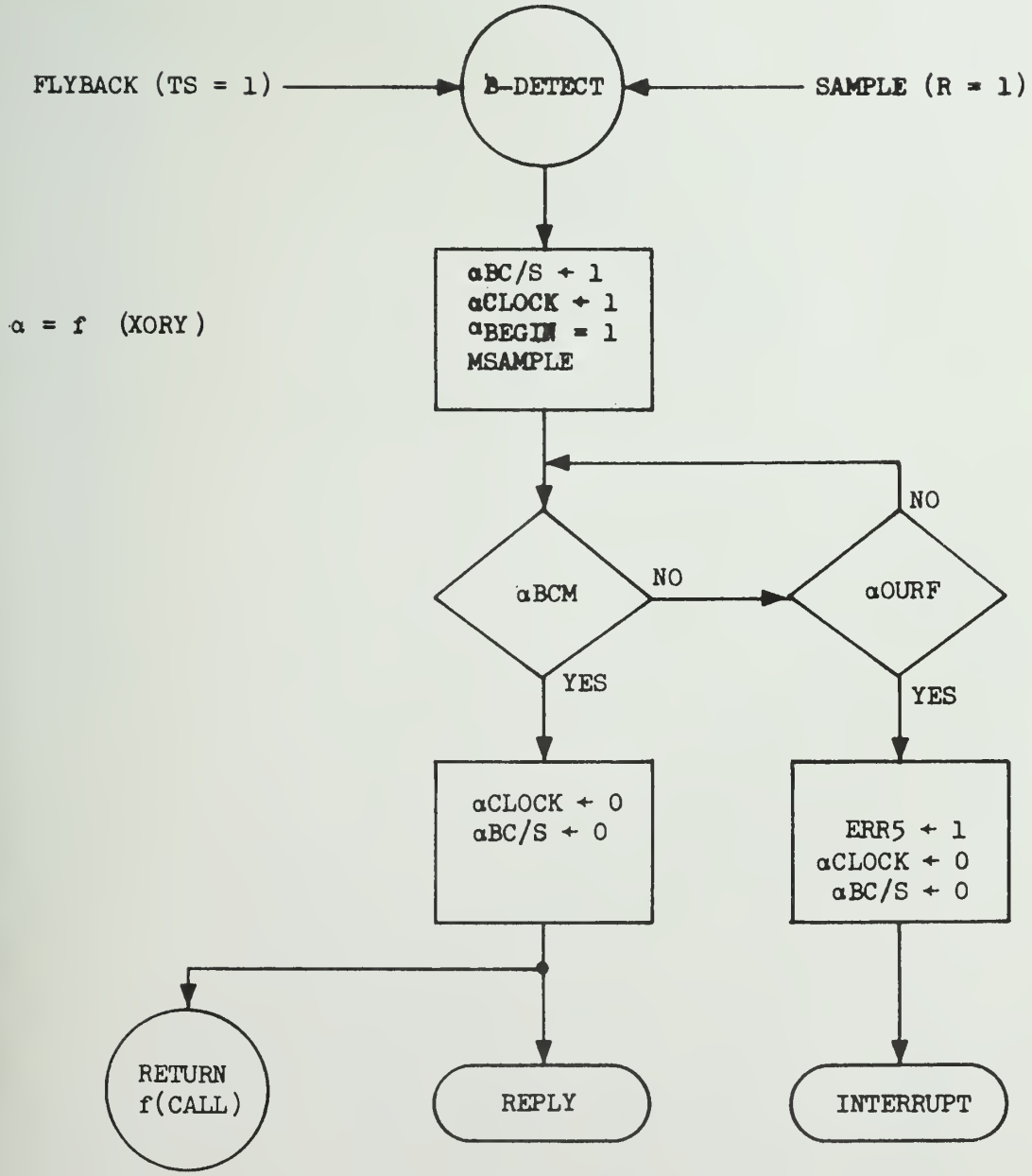


SAMPLE (INCREMENT ROUTINE)

Logic Drawing of SAMPLE

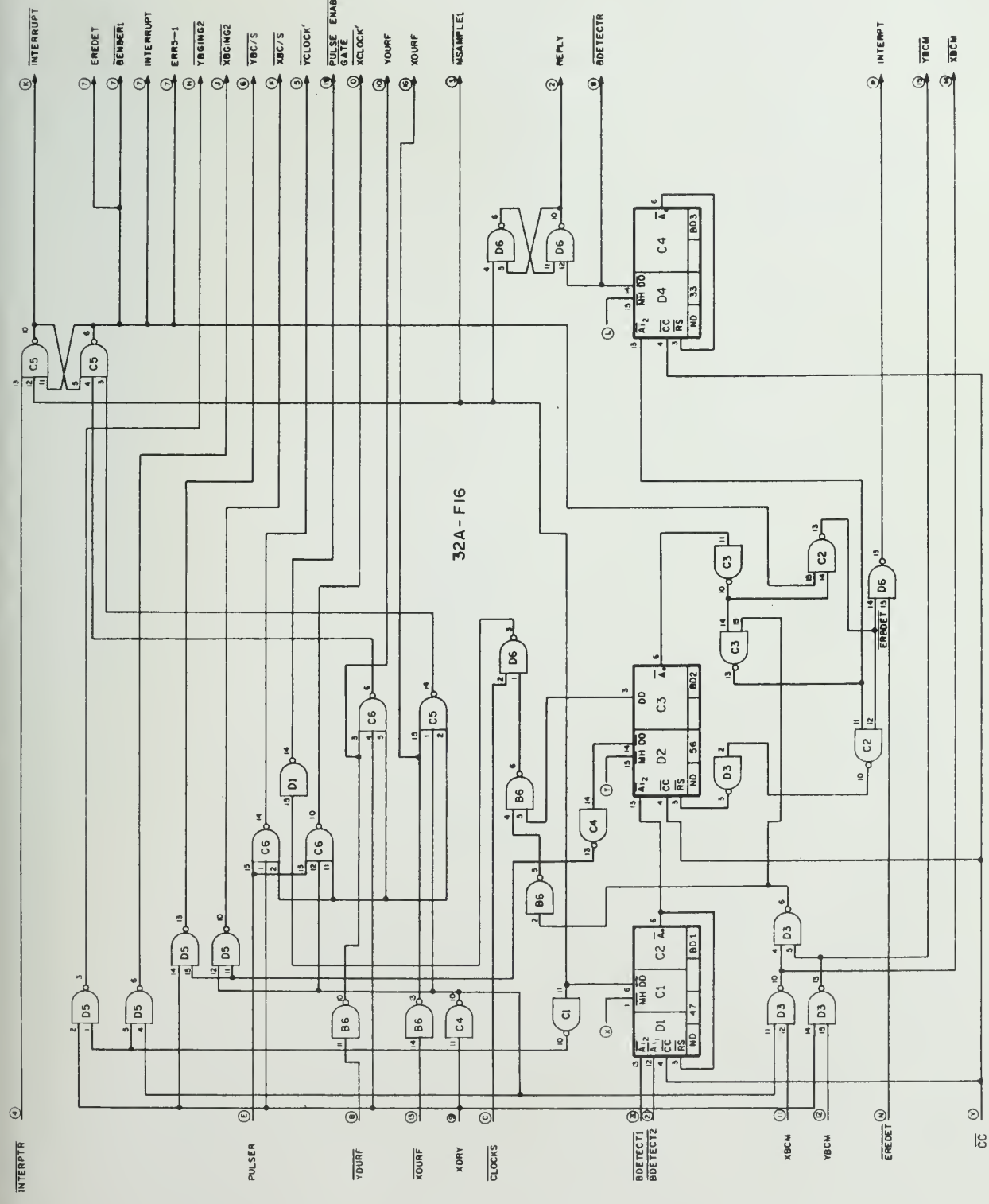


Flow Chart of B-DETECT

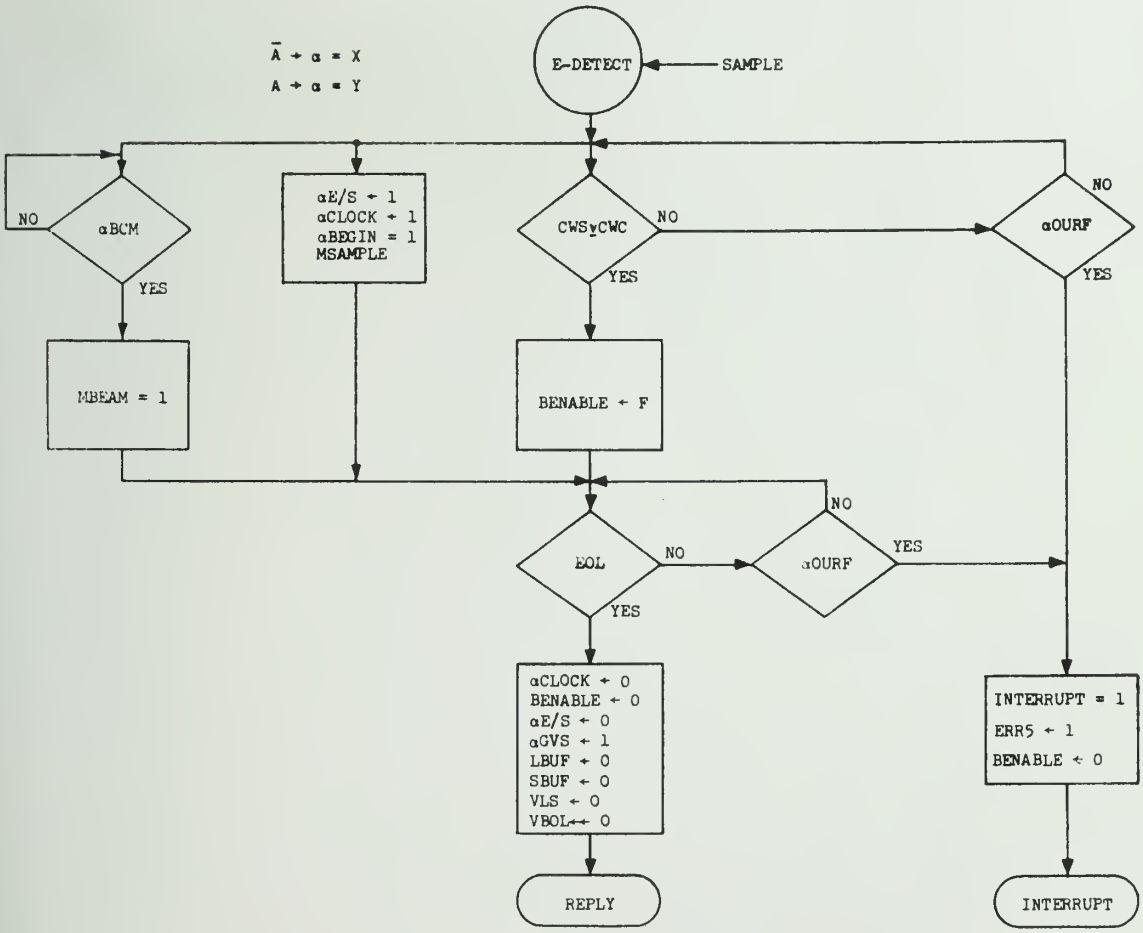


B-DETECT ROUTINE

Logic Drawing of B-DETECT

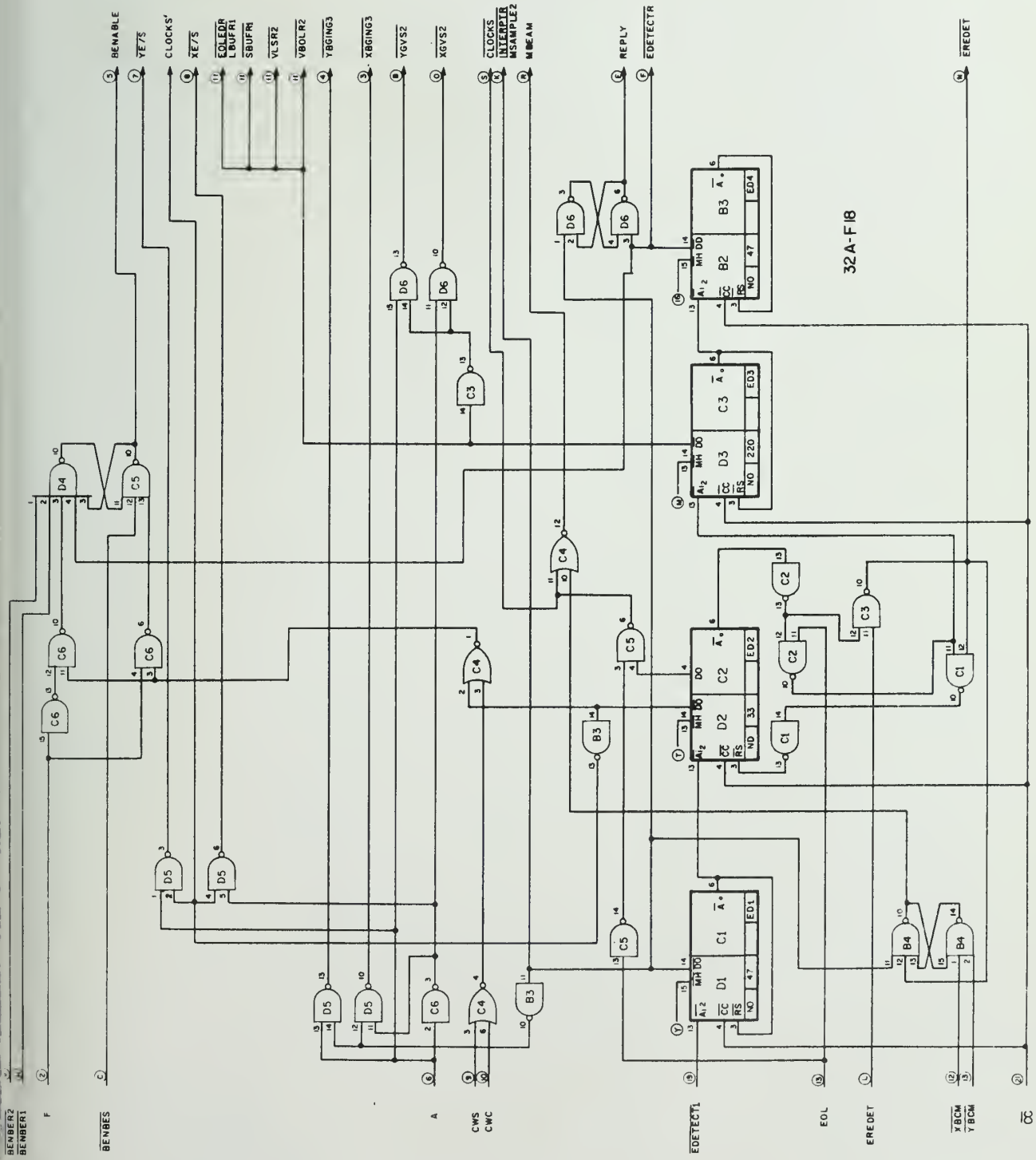


Flow Chart of E-DETECT

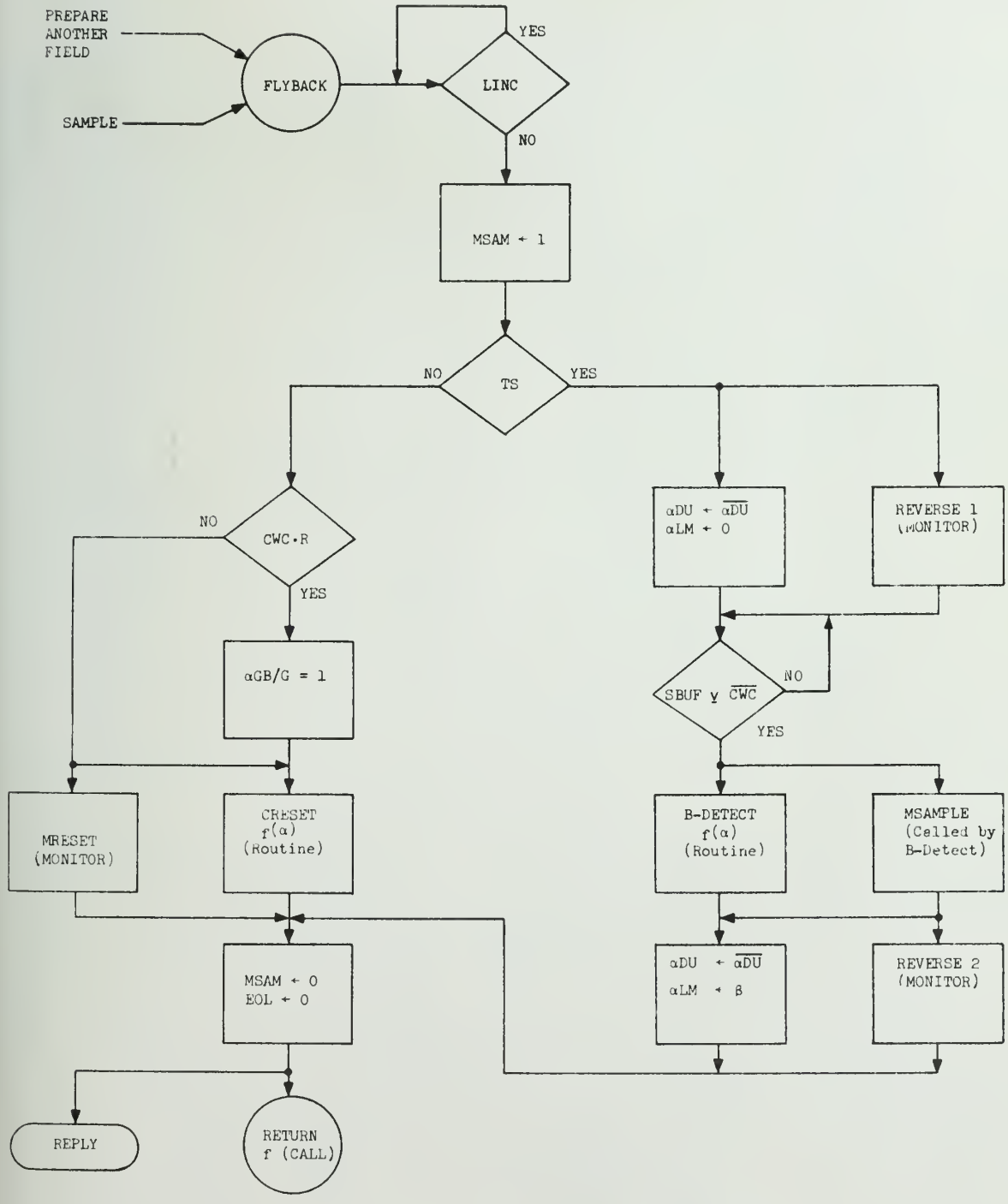


E-DETECT ROUTINE

Logic Drawing of E-DETECT



Flow Chart of FLYBACK

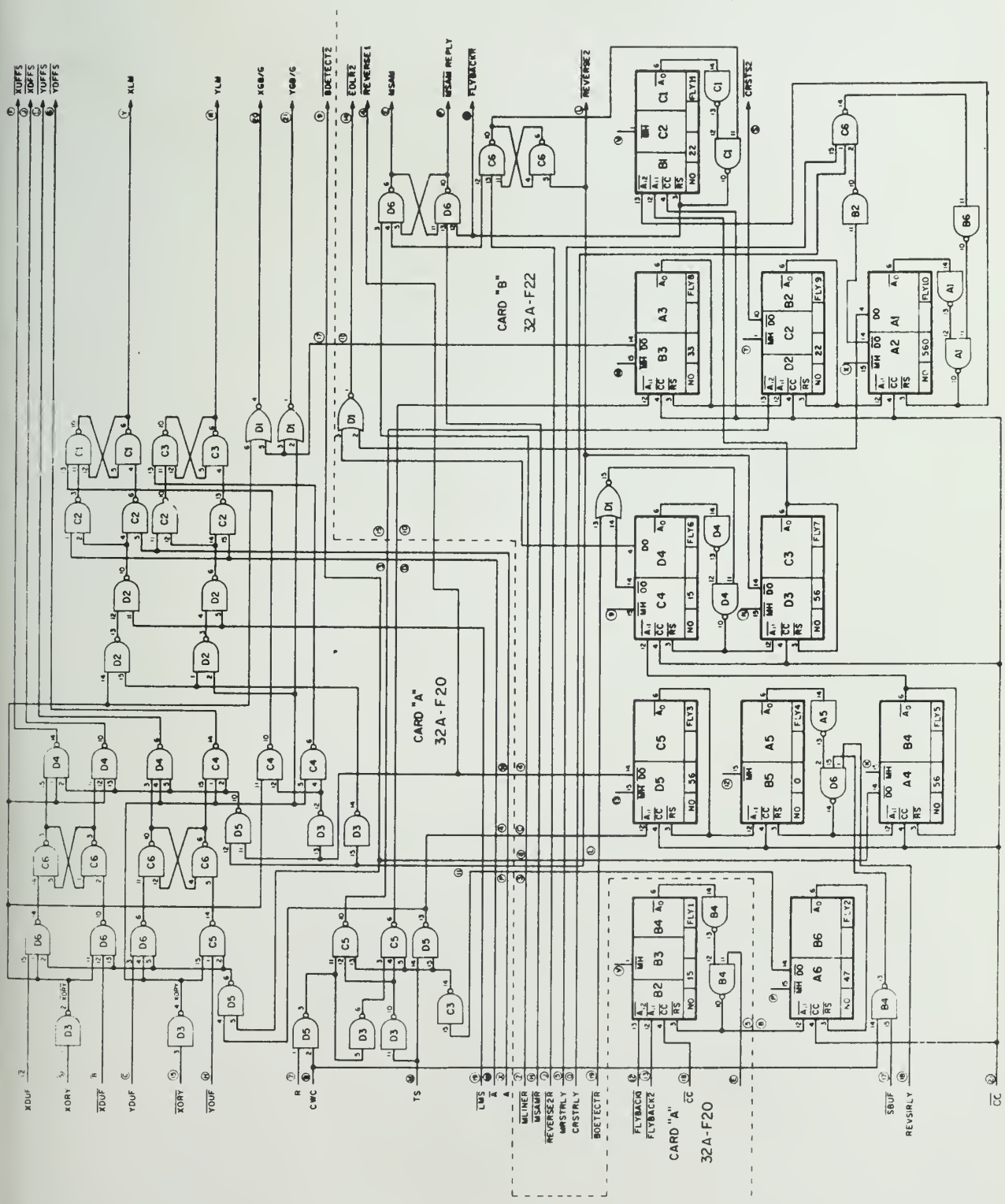


$\overline{XORY} \rightarrow \alpha = X \;\&\; \beta = A$
$XORY \rightarrow \alpha = Y \;\&\; \beta = \overline{A}$

$CWC = (DF = 1) \cdot WRITE \cdot C$
$TS = \overline{G} \cdot (h = 0) \cdot (D = 0,0) \cdot \overline{V}$

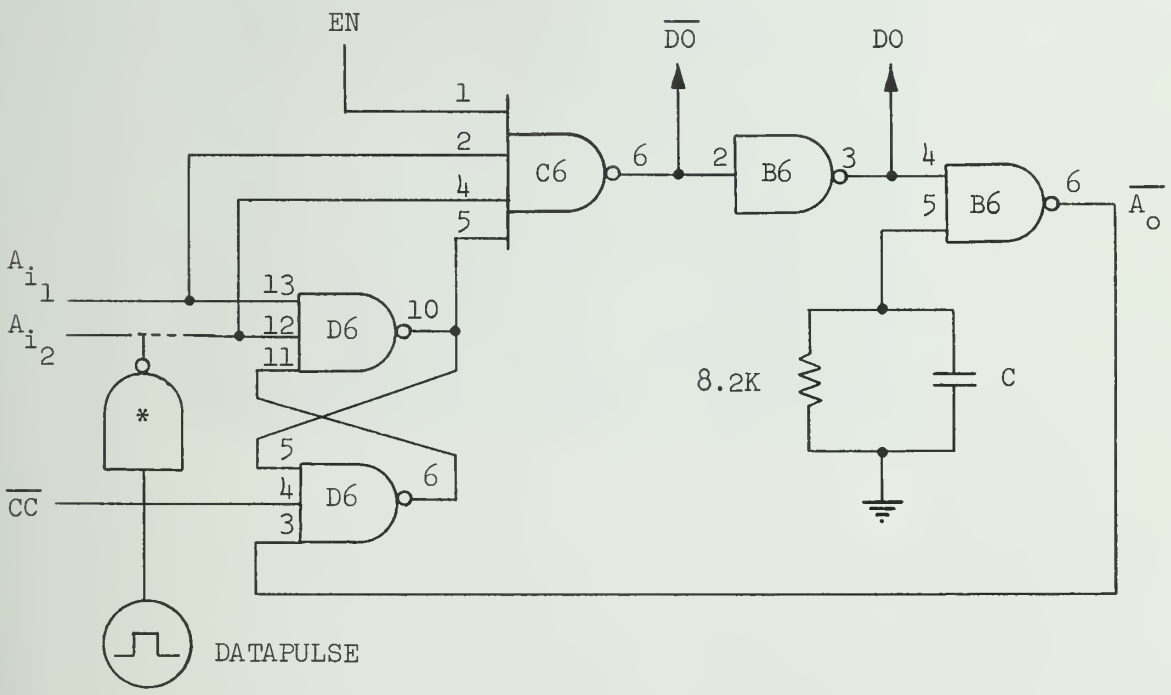
FLYBACK (Routine)

Logic Drawing of FLYBACK



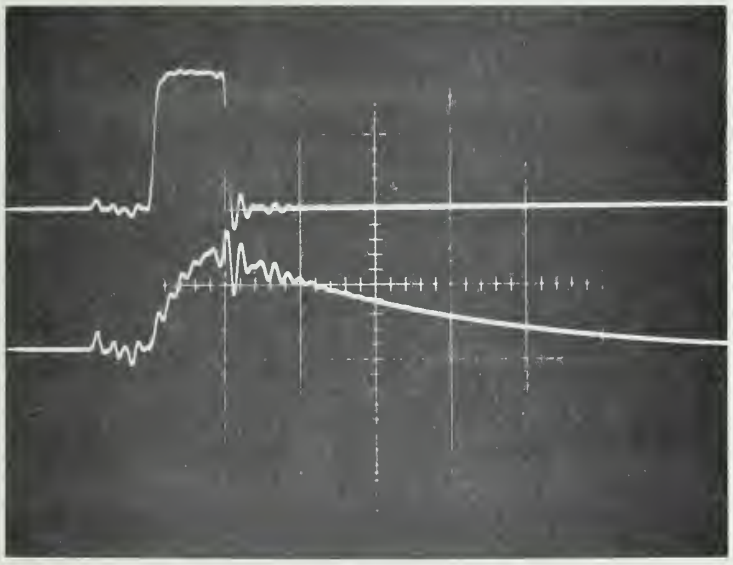
Appendix II

This appendix contains test results for a typical control point configuration. The card used in these tests was Buffer Control A (32B-M20) Serial #016170 Control Point #1



*Spare NAND used to obtain negative going pulse from DATAPULSE required to set flip-flop.

The test results and photographs in this appendix all refer to the above circuit configuration.



SCOPE SETTINGS

TOP TRACE: 2V/DIV.

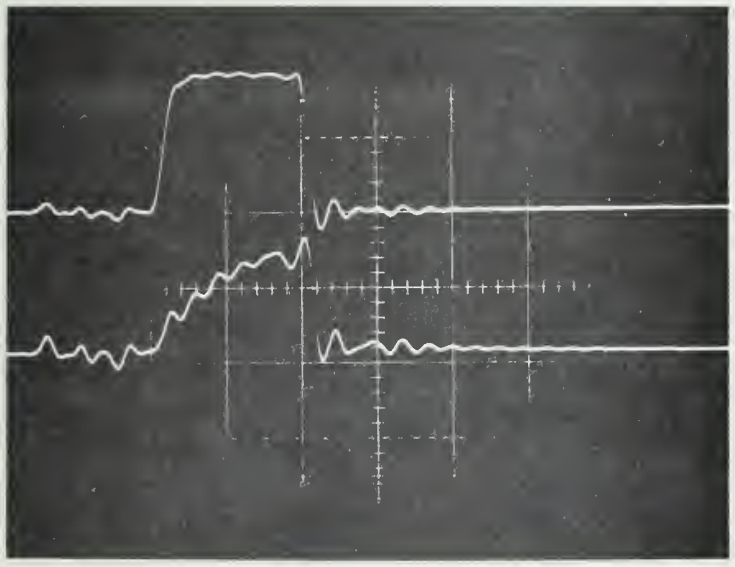
BOTTOM TRACE; IV/DIV.

SWEEP SPEED: 100nS/DIV.

IDENTIFICATION

TOP TRACE: Input pulse at pin 4 of B6

BOTTOM TRACE: PIN 5 of B6 - Charge & discharge curve of delay element (22 pfd. & 8.2K) without hot carrier diode.



SCOPE SETTINGS

TOP TRACE: 2V/DIV.

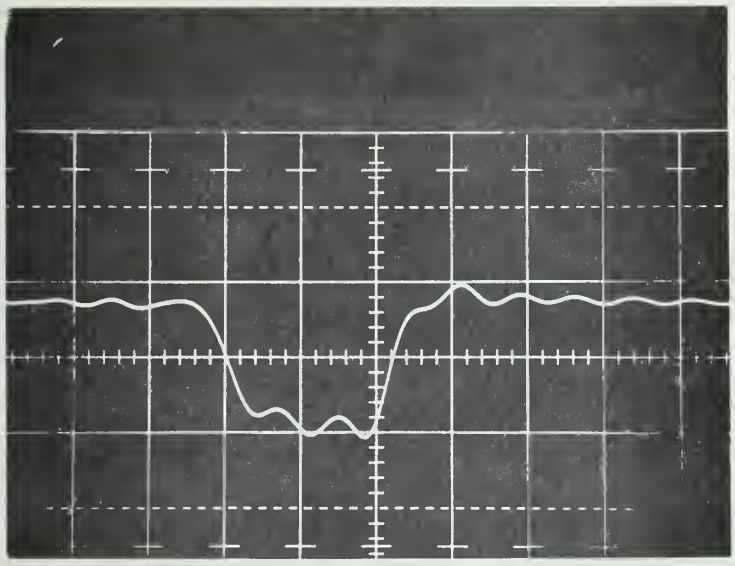
BOTTOM TRACE: 1V/DIV.

SWEEP SPEED: 50nS/DIV.

IDENTIFICATION

TOP TRACE: Input pulse at pin 4 of B6

BOTTOM TRACE: Pin 5 of B6 - Change & discharge curve of delay elements (22 pfd. & 8.2K) with hot carrier diode



SCOPE SETTING

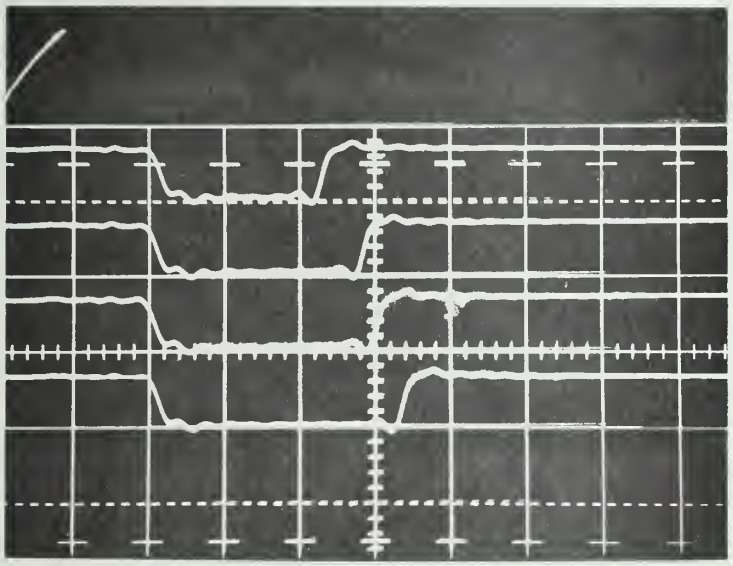
TRACE: 2V/DIV.

SWEEP SPEED: 20nS/DIV.

IDENTIFICATION

TRACE: Output pulse at pin 6 of C6 ($\overline{DO}$) using no delay elements (R & C). Pulse width indicates the inherent delay of control point itself, consisting of collector delays, wiring capacitance, etc.

NOTE: Typical control point cycle time is therefore approximately 20 MHz



SCOPE SETTINGS

ALL TRACES: 5V/DIV.

SWEEP SPEED: 50nS/DIV.

IDENTIFICATION

TRACES: Output pulse at pin 6 of C6 ($\overline{DO}$) using various values of capacitance in the timing element and R equal to 8.2K in all cases.

		C (pfd.)
TOP TRACE	#1	22
	#2	47
	#3	68
BOTTOM	#4	100

AEC-427
(6/68)
CM 3201

U. S. ATOMIC ENERGY COMMISSION

UNIVERSITY-TYPE CONTRACTOR'S RECOMMENDATION FOR DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

EC REPORT NO.	2. TITLE
eport No. 400 00-1018-1206	STANDARDIZATION OF CONTROL POINT REALIZATION

YPE OF DOCUMENT (Check one):

- [X] a. Scientific and technical report
- [ ] b. Conference paper not to be published in a journal:
  - Title of conference _____
  - Date of conference _____
  - Exact location of conference _____
  - Sponsoring organization _____
- [ ] c. Other (Specify) _____

RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

- [X] a. AEC's normal announcement and distribution procedures may be followed.
- [ ] b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.
- [ ] c. Make no announcement or distrubution.

REASON FOR RECOMMENDED RESTRICTIONS:

SUBMITTED BY: NAME AND POSITION (Please print or type)

Ronald Gustav Martin, Research Engineer

Organization

Department of Computer Science
University of Illinois
Urbana, Illinois 61820

Signature	Date
Bruce H. McCormick	May 21, 1970

FOR AEC USE ONLY

AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

PATENT CLEARANCE:

- [ ] a. AEC patent clearance has been granted by responsible AEC patent group.
- [ ] b. Report has been sent to responsible AEC patent group for clearance.
- [ ] c. Patent clearance not required.